Lecture 3: Packages and RMarkdown

UNIVERSITY OF
SAN FRANCISCO

James D. Wilson

BSDS 100 - Intro to Data Science with R

- Packages in R

- RMarkdown

  - Installation and Use

  - The Markdown language

Part I: Packages in R

Packages in R

- Vanilla R comes with extensive capabilities

  ...BUT...

- some of the most exciting features in R are available as optional modules called Packages that you can download and install

- Like R, packages are free, user-contributed modules that you can create, download and install

- As of June 2016, there were $\sim 7,000$ R packages!

What are R Packages?

- Packages are collections of functions, data and compiled code in a well-defined format

- **Caution!** Given R packages are user-contributed, some contain errors, so feel free to use R for analysis, but maybe double-check your output before you buy or sell billions of dollars of stock based on R package calculations

Basic Package functions

Function	Action
`library()`	lists which packages have been *downloaded* on your current version of `R`
`library(blue)`	loads package `blue` to current environment
`require(blue)`	same as `library(blue)`
`search()`	lists which packages are loaded and ready to use in *current session*
`install.packages("blue")`	installs package `blue`

Important: You only need to download a package once, but you always need to load packages when on a new `R` session!

```
> install.packages("microbenchmark")
  % Total     % Received % Xferd  Average Speed   Time    Time     Time
                                  Dload  Upload   Total   Spent    Left
  0      0    0     0    0     0      0       0 --:--:-- --:--:-- --:--:--

The downloaded binary packages are in
  /var/folders/jm/3w7pqfms0nvg_ypvnvkkk83h0000gn/...


> microbenchmark(3^3)
Error: could not find function "microbenchmark"

# using the double colon operator allows you to access functions from
#   packages that are not loaded


> microbenchmark::microbenchmark(3^3)
Unit: nanoseconds
 expr min  lq   mean median    uq  max neval
  3^3 203 209 365.53    271 372.5 6112   100
```

```
# this loads the package
> library(microbenchmark)

> microbenchmark(3^3)
Unit: nanoseconds
 expr min  lq   mean median  uq  max neval
  3^3 156 159 311.88    226 312 5372   100
```

Maintaining R Packages

- Packages are often updated by their authors, so be sure to keep your packages up to date

  - `update.packages()` will update all packages you've downloaded

  - `installed.packages()` will list all packages you have downloaded, along with their version numbers, dependencies and other information

User-friendly Package Functionality

A far less cumbersome way to install, load and update packages in RStudio is using the appropriate icons in the "Packages" window

Autoload of R Packages
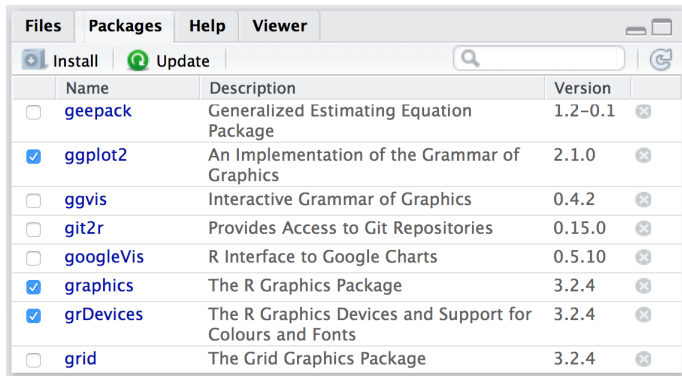
Packages sometimes have dependencies on other packages, e.g.,

```
#load the package MatchIt
> library("MatchIt", lib.loc="/Library/Frameworks/R.framework/...")
Loading required package: MASS
```

- Without asking, when loading the MatchIt package, the MASS package is automatically loaded

- This is helpful in one respect, since MatchIt leverages certain functionality from MASS; if MASS wasn't automatically loaded, then calling certain functions from MatchIt might throw an error

Caution about Masking Functions

- Be careful of *function masking*: because there are so many R packages available from so many different authors, it often happens that different packages have identically named functions

- For example, if you load package A and it contains the function foo and then load package B that contains a different function, also named foo, then calling foo will resort in the use of package B's function foo.

Searching for Functions from R Packages

When searching for a function, R searches the Global Environment first, then iterates through all packages for the function, beginning from the most recently added

```
> search()
 [1] ".GlobalEnv"        "package:reshape2"  "package:plyr"
 [4] "package:MatchIt"   "package:MASS"      "package:ggplot2"
 [7] "tools:rstudio"     "package:stats"     "package:graphics"
[10] "package:grDevices" "package:utils"     "package:datasets"
[13] "package:methods"   "Autoloads"         "package:base"

> (.packages())
 [1] "reshape2"  "plyr"      "MatchIt"   "MASS"      "ggplot2"
 [6] "graphics"  "grDevices" "utils"     "datasets"  "methods"
[11] "stats"     "base"
```

- If you are using a lot of packages and want to be certain you are calling a function from a specific package, use the double colon operator, e.g., `plyr::rename()`

- If you feel you have too many packages loaded and want to unload (detach) one, you can use the following command: `detach("package:MatchIt", unload=TRUE)`

Part II: RMarkdown

Technical Reporting & Presentation Tools

In many cases, we would like to present code and output in an easy-to-understand document.

- Many ways to do this:

    1. LaTeX
    2. .html
    3. Markdown (RMarkdown and other variations)
    4. ...

- The (arguably) most generic and flexible technical report/presentation tool is LaTeX

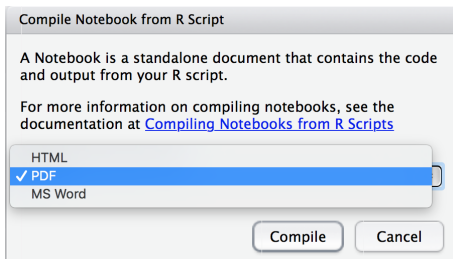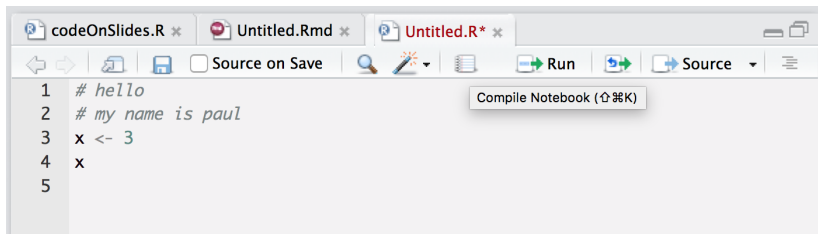- But, we'd like to deal directly with R output and code

When dealing with R code and R output, RMarkdown is a versatile and easy way to embed code and graphical output in a report or presentation

- Relies on the `knitr` package to compile and print code

- Documents are written in the Markdown language, which combines LaTeX and code

- Can directly create `.pdf` or `.html` files

- Used for presentations and for ensuring reproducibility

The Simplest ℝMarkdown Execution

The Simplest RMarkdown Execution

- `Rmd` **source file**
- `pdf` **output**

Untitled.R

Paul

Tue Jul 5 10:51:43 2016

```r
# hello
# my name is paul
x <- 3
x
```

```
## [1] 3
```

An RMarkdown Document

Create a new RMarkdown file

```
File ⇒ New File ⇒ RMarkdown...
```

We will be generating `.pdf` documents, so please be sure to install this!

Markdown Language

- RMarkdown documents are written in the Markdown language

- You can get fancy with this, but a few simple commands can bring you a long way!

- We'll review some simple rules and syntax next so you can get started.

- The header begins and ends with three dashes `--`

- There are many header options, we will examine a few basic options

```
---
title: "Untitled"
author: "James D. WIlson"
date: "January 1, 2017"
output: pdf_document
---
```

- How do you write plain text?

  ```
  Just like this
  ```

- How do you comment out a line of text?

  ```
  [//]: (comment goes here)
  <!-- (comment goes here) -->
  ```

- You can create sections / section headers using the "#" symbol

  ```
  # Header 1

  ## Header 2

  ### Header 3

  #### Header 4

  ##### Header 5

  ###### Header 6
  ```

  # Header 1
  ## Header 2
  ### Header 3
  #### Header 4
  ##### Header 5
  ###### Header 6

- Inline equations are identical to LATEXsyntax, but only *some* of the syntax is available

    See the LATEXcheatsheet here for syntax:
    https://wch.github.io/latexsheet/latexsheet.pdf

    Example: $e^{i\pi} - 1 = 0$ is written `$e^{i \pi} - 1 = 0$`

- Bold and italicized statements are written using

    - `**myBoldText**` and `*myItalicizedText*`

- In-line code that is **not** executed can be included in backticks

    CODE: To assign a value to a variable: 'myVar <- 1'

- In-line code that **is** executed can be included as 'r <insert code here>'

    CODE: Calculate 2 + 3 and print to document: 'r sum(c(2, 3))'

- At the heart of RMarkdown are code chunks, which allow for great flexibility when including raw code as well as results, from simple computations to complex graphs and analyses.
  Example:

```
```{r <sectionTitle>, <options>}
<inculde code here>
```
```

- Use `ctrl + option + I` as a shortcut to include code chunk
- `<sectionTitle>` is the *unique* name of the code chunk
- `<options>` are a sequence of options separated by commas

  Note: all labels and code chunk options must be on the same line

- `eval`: whether or not to run the code chunk

- `echo`: whether or not to include code in output document

- `include`: when `FALSE`, the code chunk is evaluated, but the results are not included in the output document

- `tidy`: whether or not to have `knitr` format printed code chunks

RMarkdown: Selected Global Chunk Options

- To set global options for all code chunks, include the following code chunk after the header

```{r <sectionTitle>, include = FALSE}
knitr::opts_chunk$set(<options>)
```

- `include = FALSE` or `echo = FALSE` is included so that the code chunk is not printed to the output document

- To have all code chunks in an RMarkdown document be suppressed, include all code in the output document

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

Including Non-R Code in Code Chunks

- RMarkdown is not limited to R code

- knitr can run code from a variety of other languages, including Python, Ruby and Bash

- To include non-R code in a code chunk, set the engine code chunk to tell knitr which language you are using

- **Example**: To include Python code, use

  ```
  ```{r engine = 'python'}
  print "Hello World"
  ```
  ```

- Additional non-R programming language interpretation is available using the highlighter package

- "Reproducible Research with R and RStudio" by Christopher Gandrud

    - This is a less technical, more pragmatic approach to RMarkdown

- "Dynamic Documents with R and knitr" by Yihui Xie

    - A more technical, detailed and rigorous treatment of RMarkdown and knitr

Quick RMarkdown Resources

- RMarkdown Cheat Sheet

  http://www.rstudio.com/wp-content/uploads/2015/02/
  rmarkdown-cheatsheet.pdf

- RMarkdown Reference Guide

  http://www.rstudio.com/wp-content/uploads/2015/03/
  rmarkdown-reference.pdf

- RMarkdown PDF Documents: Overview

  http://rmarkdown.rstudio.com/pdf_document_format.html

IMPORTANT: Style Guide

Now that Introduction to RMarkdown is complete, be sure to **thoroughly read** the Style Guide (Chapter 5), in Hadley Wickham's `Advanced R`. You will be held to that standard in your coding style moving forward.

http://adv-r.had.co.nz/Style.html

Assignment 3

- Make sure that you have *MacTex* or some other tex editor installed on your computer so that `knitr` can be used to make `.pdf` documents.

- Run this simple "Hello World" script in a new `.Rmd` file, replacing the date and name with the appropriate date and name. Create a `.pdf` file with the output.

- Complete the Computational Assignment here.

    Due: Next Tuesday at the beginning of class.