R for MEDSL Data

James Dunham

July 10, 2018

Installing R

- Install R from https://cloud.r-project.org
- Also install RStudio, an interface for working in R: https://www.rstudio.com/products/rstudio/download

Writing R code (R4DS)

```r
1 / 200 * 30
```

```
## [1] 0.15
```

```r
(59 + 73 + 2) / 3
```

```
## [1] 44.66667
```

```r
sin(pi / 2)
```

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

Assignment (R4DS)

Create new objects with <-:

```
x <- 3 * 4
x
```

```
## [1] 12
```

All R statements where you create objects have the same form:

```
object_name <- value
```

Errata:

- Shortcut in RStudio for typing <- is ALT-minus
- = is an alternative to <-. It's either convenient and the universal assignment operator or *dangerous* and *lazy*.
- -> also exists, because R.

Functions (R4DS)

R has a large [*ed:* bloated] collection of built-in functions.

Called like this:

```
function_name(arg1 = val1, arg2 = val2, ...)
```

For example:

```r
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Assign the *return value*:

```r
x <- seq(1, 10)
```

Getting help

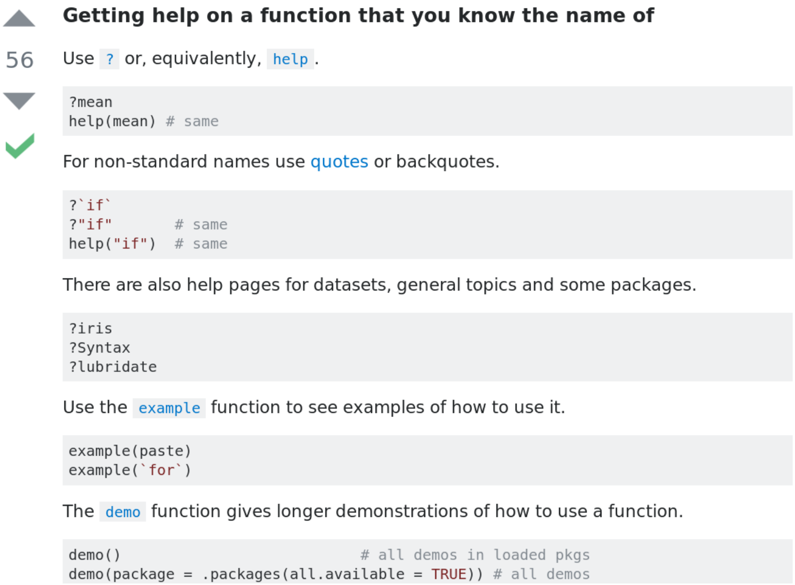
Function help:

```
?seq
help(seq)
```

Other sources of help:

- rdocumentation.org
- stackoverflow.com
- #r on Slack

StackOverflow



Getting help on a function that you know the name of

Use `?` or, equivalently, `help`.

```
?mean
help(mean) # same
```

For non-standard names use quotes or backquotes.

```
?`if`
?"if"       # same
help("if")  # same
```

There are also help pages for datasets, general topics and some packages.

```
?iris
?Syntax
?lubridate
```

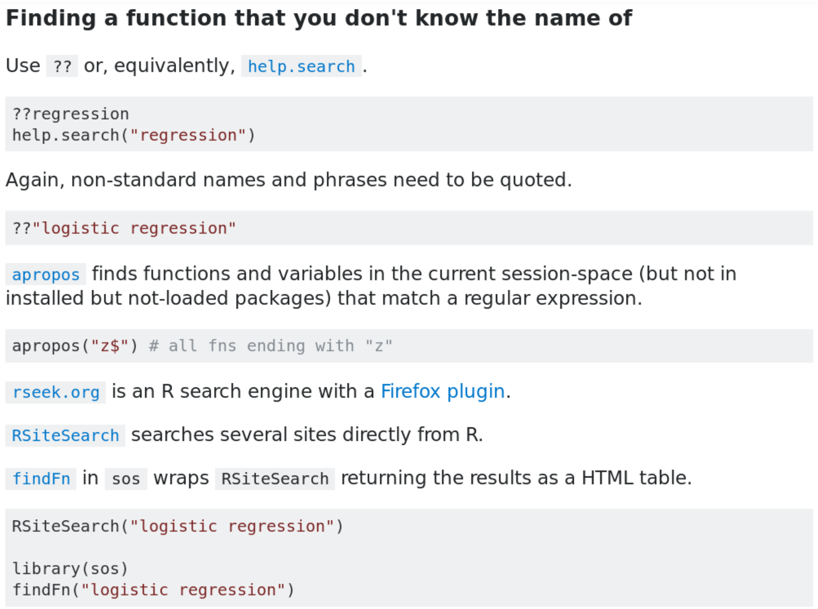
Use the `example` function to see examples of how to use it.

```
example(paste)
example(`for`)
```

The `demo` function gives longer demonstrations of how to use a function.

```
demo()                                # all demos in loaded pkgs
demo(package = .packages(all.available = TRUE)) # all demos
```

StackOverflow



Finding a function that you don't know the name of

Use `??` or, equivalently, `help.search`.

```
??regression
help.search("regression")
```

Again, non-standard names and phrases need to be quoted.

```
??"logistic regression"
```

`apropos` finds functions and variables in the current session-space (but not in installed but not-loaded packages) that match a regular expression.

```
apropos("z$") # all fns ending with "z"
```

`rseek.org` is an R search engine with a Firefox plugin.

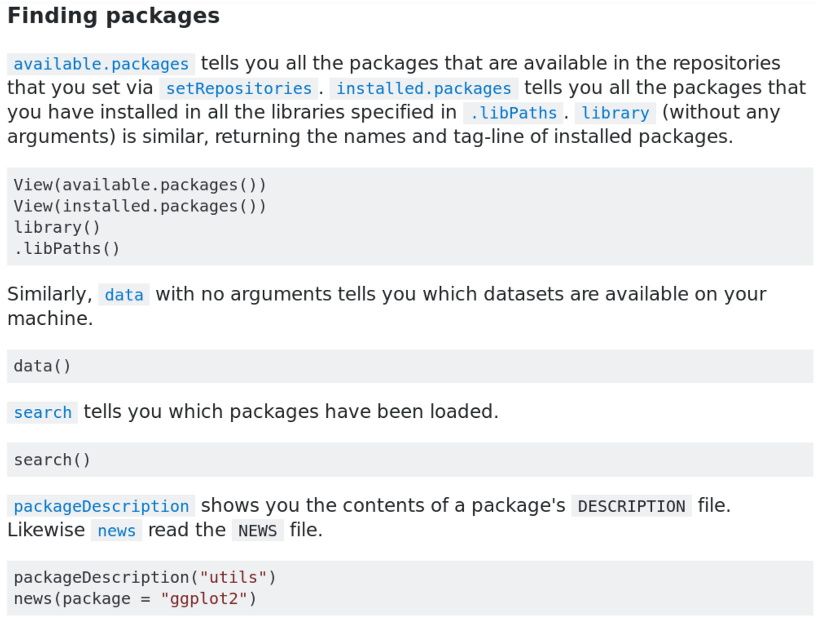
`RSiteSearch` searches several sites directly from R.

`findFn` in `sos` wraps `RSiteSearch` returning the results as a HTML table.

```
RSiteSearch("logistic regression")

library(sos)
findFn("logistic regression")
```

StackOverflow



Finding packages

`available.packages` tells you all the packages that are available in the repositories that you set via `setRepositories`. `installed.packages` tells you all the packages that you have installed in all the libraries specified in `.libPaths`. `library` (without any arguments) is similar, returning the names and tag-line of installed packages.

```
View(available.packages())
View(installed.packages())
library()
.libPaths()
```

Similarly, `data` with no arguments tells you which datasets are available on your machine.

```
data()
```

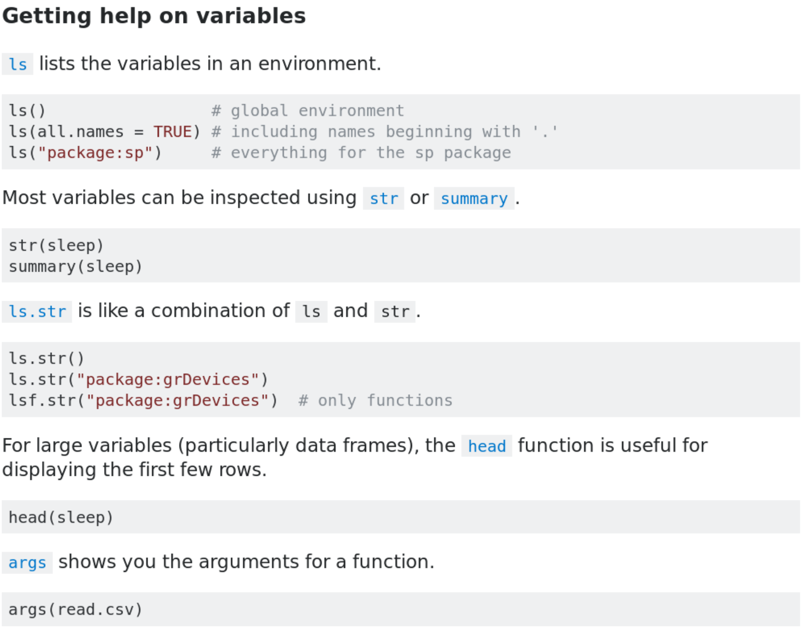
`search` tells you which packages have been loaded.

```
search()
```

`packageDescription` shows you the contents of a package's `DESCRIPTION` file. Likewise `news` read the `NEWS` file.

```
packageDescription("utils")
news(package = "ggplot2")
```

StackOverflow



Getting help on variables

`ls` lists the variables in an environment.

```
ls()                 # global environment
ls(all.names = TRUE) # including names beginning with '.'
ls("package:sp")     # everything for the sp package
```

Most variables can be inspected using `str` or `summary`.

```
str(sleep)
summary(sleep)
```

`ls.str` is like a combination of `ls` and `str`.

```
ls.str()
ls.str("package:grDevices")
lsf.str("package:grDevices")  # only functions
```

For large variables (particularly data frames), the `head` function is useful for displaying the first few rows.

```
head(sleep)
```

`args` shows you the arguments for a function.

```
args(read.csv)
```

The hadleyverse (tidyverse)

```
# https://www.tidyverse.org/
install.packages("tidyverse")
library(tidyverse)
```

A grammar for data

```r
data('mpg', package = 'ggplot2')
mpg %>%
  select(manufacturer, model, displ) %>%
  head(3)
```

```
## # A tibble: 3 x 3
##   manufacturer model displ
##   <chr>        <chr> <dbl>
## 1 audi         a4      1.8
## 2 audi         a4      1.8
## 3 audi         a4      2
```

A grammar for data

```r
mpg %>%
  select(manufacturer, model, displ) %>%
  filter(displ > 2) %>%
  head(3)
```

```
## # A tibble: 3 x 3
##   manufacturer model displ
##   <chr>        <chr> <dbl>
## 1 audi         a4      2.8
## 2 audi         a4      2.8
## 3 audi         a4      3.1
```

A grammar for data

```r
mpg %>%
  select(manufacturer, model, displ) %>%
  filter(displ > 2) %>%
  mutate(displ_squared = displ ^ 2) %>%
  head(3)
```

```
## # A tibble: 3 x 4
##   manufacturer model displ displ_squared
##   <chr>        <chr> <dbl>         <dbl>
## 1 audi         a4      2.8          7.84
## 2 audi         a4      2.8          7.84
## 3 audi         a4      3.1          9.61
```

Resources

RStudio Cheatsheets:

- “Data Import”
- “Data Transformation”
- “Work with Strings”
- “RStudio”

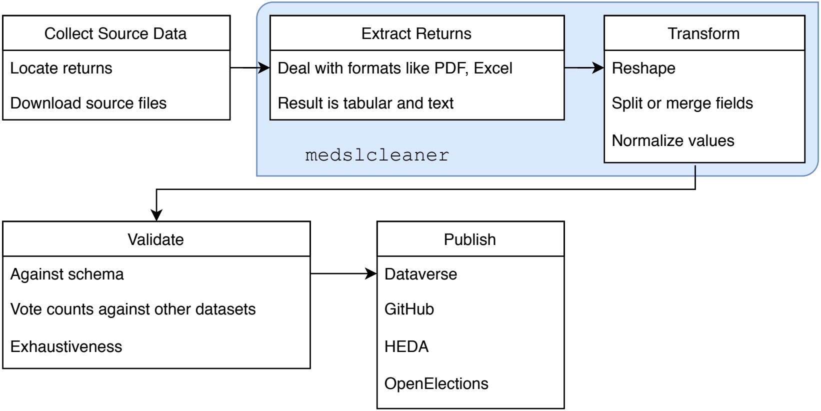
Books:

- *R for Data Science*, especially the section “Wrangle”
- *An Introduction to Statistical and Data Sciences via R*

Courses:

- DataCamp’s *Introduction to R*
- *R Basics* from Harvard’s edX Data Science Series

Workflow



Collect Source Data

Locate returns

Download source files

Extract Returns

Deal with formats like PDF, Excel

Result is tabular and text

Transform

Reshape

Split or merge fields

Normalize values

medslcleaner

Validate

Against schema

Vote counts against other datasets

Exhaustiveness

Publish

Dataverse

GitHub

HEDA

OpenElections

Our toolkit

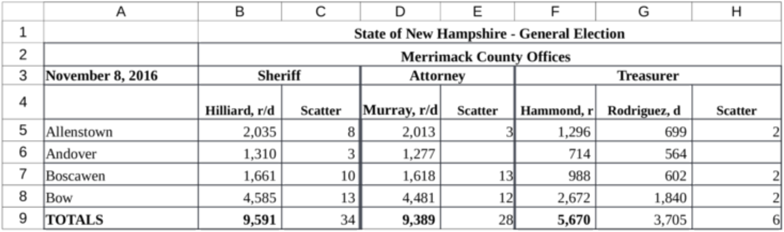
Install once:

```r
install.packages("tidyverse")
install.packages("tidyxl")
install.packages("devtools")
devtools::install_github('MEDSL/medslcleaner')
```

Load every time:

```r
library(tidyverse)
library(tidyxl)
library(medslcleaner)
```

Source data

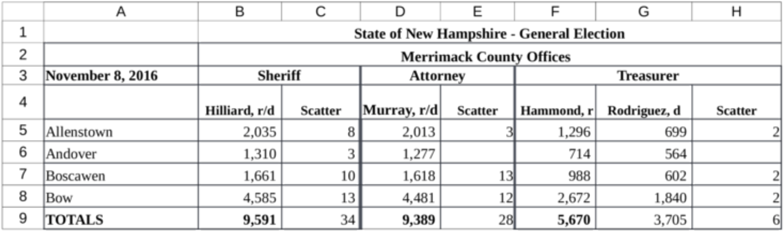


	A	B	C	D	E	F	G	H
1		State of New Hampshire - General Election						
2		Merrimack County Offices						
3	November 8, 2016	Sheriff		Attorney		Treasurer		
4		Hilliard, r/d	Scatter	Murray, r/d	Scatter	Hammond, r	Rodriguez, d	Scatter
5	Allenstown	2,035	8	2,013	3	1,296	699	2
6	Andover	1,310	3	1,277		714	564	
7	Boscawen	1,661	10	1,618	13	988	602	2
8	Bow	4,585	13	4,481	12	2,672	1,840	2
9	**TOTALS**	**9,591**	34	**9,389**	28	**5,670**	3,705	6

Multiple headers:

- Row 2: `jurisdiction`
- Row 3: `office`
- Row 4: `candidate`
- Column A: `precinct`

Source data



	A	B	C	D	E	F	G	H
1		**State of New Hampshire - General Election**						
2		**Merrimack County Offices**						
3	**November 8, 2016**	**Sheriff**		**Attorney**		**Treasurer**		
4		**Hilliard, r/d**	**Scatter**	**Murray, r/d**	**Scatter**	**Hammond, r**	**Rodriguez, d**	**Scatter**
5	Allenstown	2,035	8	2,013	3	1,296	699	2
6	Andover	1,310	3	1,277		714	564	
7	Boscawen	1,661	10	1,618	13	988	602	2
8	Bow	4,585	13	4,481	12	2,672	1,840	2
9	**TOTALS**	**9,591**	34	**9,389**	28	**5,670**	3,705	6

Multiple-column or “merged” cells:

- `Sheriff`
- `Attorney`
- `Treasurer`

Typical approach

```r
# Get the path to the packaged example
merrimack_path <- spreadsheet_example('merrimack')

# Use `read_excel` from the `readxl` package
sheet <- read_excel(merrimack_path)
```

Result

```r
sheet %>%
  # Select first three columns
  select(1:3)
```

```
## # A tibble: 8 x 3
##   ..1     `State of New Hampshire - General Election` ..3
##   <chr>     <chr>                               <chr>
## 1 <NA>     Merrimack County Offices             <NA>
## 2 42682     Sheriff                             <NA>
## 3 <NA>     Hilliard, r/d                      Scatter
## 4 Allenstown 2035                                   8
## 5 Andover    1310                                   3
## 6 Boscawen   1661                                  10
## 7 Bow       4585                                   13
## 8 TOTALS     9591                                  34
```

Alternative approach

Using `medslcleaner` and `tidyxl`,

- Identify which cells are *data* and which are *headers*
- Define the relationships between data cells and header cells

Reading from the disk

Instead of `read_excel`,

```
cells <- read_xlreturns(merrimack_path)
```

```
# Take a look at a few rows and columns
peek <- cells %>%
  select(-sheet) %>%
  filter(data_type != "blank") %>%
  filter(row > 3)
```

Result

```r
head(peek)
```

```
##   address row col data_type         value
## 1      B4   4   2 character Hilliard, r/d
## 2      C4   4   3 character       Scatter
## 3      D4   4   4 character   Murray, r/d
## 4      E4   4   5 character       Scatter
## 5      F4   4   6 character    Hammond, r
## 6      G4   4   7 character  Rodriguez, d
```

- Each row gives the contents of a single spreadsheet cell;
- Columns `row` and `col` give the cell's position;
- Excel identifies columns with letters, but we're using numbers.

Associating headers and data

Consider again the precinct names in column 1 of the Merrimack spreadsheet.

```
cells %>%
  select(-sheet) %>%
  filter(col == 1 & row > 4)
```

```
##   address row col data_type      value
## 1      A5   5   1 character Allenstown
## 2      A6   6   1 character    Andover
## 3      A7   7   1 character   Boscawen
## 4      A8   8   1 character        Bow
## 5      A9   9   1 character     TOTALS
```

Associating headers and data

To associate each `precinct` header with all the cells to their right:

```r
cells <- cells %>%
  as_header('precinct', cols = 1, right = TRUE)
```

- We just created a new variable `precinct`
- It takes as values the contents of cells where `col` is `1` (otherwise `NA`) …
- For all the cells to the `right` of the header cells in the spreadsheet

Associating headers and data

The result:

```
cells %>%
  filter(row > 4) %>%
  select(-sheet) %>%
  head()
```

```
##   address row col data_type value   precinct
## 1      B5   5   2   numeric  2035 Allenstown
## 2      C5   5   3   numeric     8 Allenstown
## 3      D5   5   4   numeric  2013 Allenstown
## 4      E5   5   5   numeric     3 Allenstown
## 5      F5   5   6   numeric  1296 Allenstown
## 6      G5   5   7   numeric   699 Allenstown
```

as_header

```
function (.data, idcol, rows = TRUE, cols = TRUE,
  right = FALSE,  down = FALSE, .drop = TRUE)
```

Identifying headers:

- Arguments `row` and `col` select header cells by spreadsheet row and column indexes
- If we specify `rows = 2`, values of the second row in the spreadsheet would be considered header values
- With both `rows = 2` and `cols = 1`, we could define the cell in the second row and first column as a header cell
- In more difficult cases, we can use logical functions for selection

as_header

```
function (.data, idcol, rows = TRUE, cols = TRUE,
  right = FALSE,  down = FALSE, .drop = TRUE)
```

Identifying data:

- Identify data cells by giving directions from header cells
- We can move rightward, downward, or both

as_header

```
function (.data, idcol, rows = TRUE, cols = TRUE,
  right = FALSE,  down = FALSE, .drop = TRUE)
```

Argument .drop:

- Defining cells as headers drops them from the data after moving their values into a new column
- One way to think of the as_header function is as transformation of headers from spreadsheet cells into characteristics of spreadsheet cells.

Associating headers and data

Let's do the remaining identifiers:

```r
cells <- cells %>%
  as_header('jurisdiction', rows = 2, cols = 2,
    down = TRUE, right = TRUE) %>%
  as_header('office', rows = 3, right = TRUE,
    down = TRUE) %>%
  as_header('candidate', rows = 4, down = TRUE)
```

Associating headers and data

Finally, to keep only the columns we created and rename the value column votes:

```
cells <- cells %>%
  filter(row > 4) %>%
  finalize()

glimpse(cells)
```

```
## Observations: 35
## Variables: 5
## $ votes        <int> 2035, 8, 2013, 3, 1296, 699, 2, 1310, 3,
## $ precinct     <chr> "Allenstown", "Allenstown", "Allenstow
## $ jurisdiction <chr> "Merrimack County Offices  ", "Merrima
## $ office       <chr> "Sheriff", "Sheriff", "Attorney", "Atto
## $ candidate    <chr> "Hilliard, r/d", "Scatter", "Murray, r
```

Full Solution

```r
cells <- read_xlreturns(merrimack_path)

cells <- cells %>%
  as_header('jurisdiction', rows = 2, cols = 2,
    down = TRUE, right = TRUE) %>%
  as_header('precinct', cols = 1, right = TRUE) %>%
  as_header('office', rows = 3, right = TRUE,
    down = TRUE) %>%
  as_header('candidate', rows = 4, down = TRUE)

# Drop remaining header rows and `finalize`
cells <- cells %>%
  filter(row > 4 & col> 1) %>%
  finalize()
```

Validation

Schema define our expectations about data:

```
- name: votes
  title: Vote Count
  description: Number of votes received.
  source: Precinct returns for `jurisdiction`.
  type: integer
  constraints:
    required: true
```

Representation in R

```r
data(fields, package = 'medslcleaner')
str(fields[['votes']])
```

```
## List of 6
##  $ name       : chr "votes"
##  $ title      : chr "Vote Count"
##  $ description: chr "Number of votes received."
##  $ source     : chr "Precinct returns for `jurisdiction`."
##  $ type       : chr "integer"
##  $ constraints:List of 1
##   ..$ required: logi TRUE
```

Validation

```r
data(wyoming, package = 'medslcleaner')
wyoming %>%
  mutate(precinct = substr(precinct, 1, 10)) %>%
  select(state_postal, jurisdiction, precinct,
    office, candidate, writein, votes) %>%
  head()
```

```
##   state_postal jurisdiction   precinct   office  candidate w
## 1           WY       Albany Shields St US House [Write-in]    TRU
## 2           WY       Albany Albany Cou US House [Write-in]    TRU
## 3           WY       Albany Harmony Sc US House [Write-in]    TRU
## 4           WY       Albany Centennial US House [Write-in]    TRU
## 5           WY       Albany Rock River US House [Write-in]    TRU
## 6           WY       Albany Shields St US House [Write-in]    TRU
```

Validation

```
validate(wyoming)
## Validating:
##   year
##   state_postal
##   jurisdiction
##   precinct
##   office
##   district
##   stage
##   special
##   candidate
##   writein
##   party
##   mode
##   votes
##   dataverse
## Success!
```

Validation

```
returns <- data.frame(votes = c(2, NA))
returns
```

```
##   votes
## 1     2
## 2    NA
```

```
validate_field(returns, 'votes')
```

```
#> Error: votes has missing values.
```

Validation

```
select_missing(returns, 'votes')
```

```
## 1/2 rows have missing "votes" values
```

```
##    votes
## 1:    NA
```

```
validate(returns)
```

```
#> Error: .data does not have name year
```

Spreadsheet resources

- medslcleaner documentation
- tidyxl documentation
- *Spreadsheet Munging Strategies*