Session 2: Data Exploration

Clarke Iakovakis
Scholarly Communications Librarian
University of Houston-Clear Lake
About me

Scholarly Communications Librarian at the University of Houston-Clear Lake
Outline for Today

• R Packages
• Getting data into R
• Exploring your data
This will (still) be overwhelming!

Refer to the handouts and help resources provided in Session 1 and stick with it!
Users write their own function libraries and release them as R Packages.

BenM
@benmiller314

me on python: spend 2 hrs working out how to do X. discover library designed to do X better. spend 3 hrs wading into that library. repeat.

10:54 AM - 20 Jun 2017

https://twitter.com/benmiller314/status/877232360265615360
Packages

- To install
  - Click on Packages > Install
  - Or type directly in the console:

```r
> install.packages("dplyr")
```
Install this helpful package of packages:

> install.packages("tidyverse")

The tidyverse is "a coherent system of packages for data manipulation, exploration and visualization that share a common design philosophy"

Are you getting error message 
Warning: unable to move temporary installation?
It’s possibly due to antivirus protection. Install directly from R (as opposed to R Studio) or see
• [https://stackoverflow.com/a/44256520](https://stackoverflow.com/a/44256520)
• [https://stackoverflow.com/a/23167680](https://stackoverflow.com/a/23167680)
Installing **tidyverse** giving you trouble? Just install these for now:

```r
> install.packages("dplyr")
> install.packages("readxl")
> install.packages("stringr")
> install.packages("ggplot2")
```
Packages

After installing a package, you must load it into your R session

```r
> select(books, SUBJECT)
Error: could not find function "select"
```

```r
> library("tidyverse")
```
Reading data into R

• When you **read** a file into R, you must save it as an **object**. In other words, create a copy of it in R’s working memory.

• When you manipulate the data, you are manipulating that object in the R environment, **not the data file itself**.

• You have to **write** the data frame to disk if you want to create an actual file; e.g. use functions `write.table()` or `write.csv()`
Getting data into R

- Manually create it
- Import it from a file
  - Text file: TXT, CSV, TSV
  - Excel: XLSX
  - Statistics program: SPSS, MATLAB, Stata
  - Databases: MySQL
  - JSON, HTML, XML
- Gather it from the web
  - Connect to webpages, servers, or APIs
  - Webscraping

Workflow image from Garrett Grolemund & Hadley Wickham, *R for Data Science*. http://r4ds.had.co.nz/explore-intro.html. Licensed under CC BY-NC-ND.
Set your working directory

The **working directory** is the folder on your computer R will use for reading and writing files

```r
> setwd("C:/Users/iakovakis/Documents/ALCTS R Webinar")
```

- Put in quotation marks " "
- Use forward slash: /
Set your working directory

You can copy/paste from directory window and do a Find \ Replace / in R Studio
Set your working directory

Print the current working directory to the console:

> getwd()
[1] "C:/Users/iakovakis/Documents/ALCTS R Webinar"
Set your working directory

You can then use \./ as the root directory

```r
> read.csv(file = ".//data/books.csv")
```

will be the same as

```r
> read.csv(file = "C:/Users/iakovakis/Documents/ALCTS R Webinar/data/books.csv")
```
Main functions for reading tabular data:

- `read.table()`: reads all text files in tabular format
- `read.csv()`: reads comma-separated values files
- `read.delim()`: reads tab-separated values files
• Text files use delimiters to separate data values
• comma separated values (CSV) use commas
• Tab separated values use tabs
• Data values are typically in quotation marks
R functions take arguments

• A function can take a specified number of arguments
• Arguments can be found on the help page for that function

> ?read.table
`read.table` arguments

- `file = "./data/books.csv"
  The path and file which data are to be read from.

- `header = TRUE`
  The file contain the names of the variables as its first line

- `sep = ",,"`
  Specify the character separating fields (columns)
`read.table` arguments

```r
colClasses = c("ISBN" = "character")

Specify the **class** of columns in the dataset

na.strings = ""

In the original dataset, how are missing values represented?

stringsAsFactors = FALSE

Should character vectors be converted to factors?

I always set this to **FALSE**
The only required argument is `read.table(file = )`

But specifying more arguments helps bring in the dataset faster and in the shape you want it in.
> books <- read.table("./books.csv"
, stringsAsFactors = F
, header = T
, sep = ","
, colClasses = c("TOT.CHKOUT" = "integer"))
Reading tabular data

R Studio has an Import Dataset tool in the Environment Pane (upper right).

It runs the `read.table()` or `read.csv()` function.
To read Microsoft Excel files, either:

a) Save your file in Excel as a CSV and load it in R with `read.csv()` or `read.table()`

b) Or, use the `readxl` package in R

```r
> install.packages("readxl")
> library("readxl")
> read_excel("./data/books.xlsx")
```
Tidying & transforming data

Program

Import → **Tidy** → Transform → Communicate

Understand

Visualise

Model
Exploring data frames

> View(books)
Exploring data frames

> dim(books)
[1] 4000 12

> nrow(books)
[1] 4000

> ncol(books)
[1] 12

Dimensions: 4,000 rows, 12 columns

Number of rows

Number of columns
Exploring data frames

> str(books)

data.frame': 4000 obs. of 12 variables:
$ callnumber : chr "SI 1.2:Af 8/3" "HC110.C6 L852 1999" "HX91.05 B57 1999" "I 20.47:P 93" ...
$ title : chr "A human ideal in African art :~Bamana figurative sculpture /" "Untold millions :
~secret truths about marketing to gay and lesbian consumers /" "Agrarian socialism in America :
~Marx, Jefferson, and Jesus in the Oklahoma countryside, 1904-1920 /" "1:100 000-
scale metric topographic map of Price, Utah, 1980 :~30 x 60 minute series (topographic) /" ...
$ author : chr "Kate Ezra." "Grant Lukenbill." "Jim Bissett." "United States Department of
the Interior, Bureau of Indian Affairs." ...
$ LOCATION : chr "clusd" "clstk" "clstk" "clusd" ...
$ TOT.CHKOUT : int 0 0 1 0 0 0 0 1 0 ...
$ LOUTDATE : chr " - - " " - - " "03-04-2015 14:53" " - - " ...
Use the dollar sign $ to specify a variable in a data frame

> str(books$LOCATION)
chr [1:4000] "clstk" "clstk" "clstk" "clstk" "clstk" "clstk"...
unique() returns all the distinct values in a variable

> unique(books$LOCATION)
[1] clcdr clchi clcir clcre clfhd clids cljre cljv clref clrsd clstk clthe cltxd clua clusd

In this case, it displays all unique book location codes.
**table()** returns frequency counts for a specified variable

```r
> table(books$LOCATION)
clcdr  clchi  clcir  clcre  clfhd  clids  cljre  cljuv  clref  clrsd  clstk  clthe  cltxd  clua  clusd
11    35     1     3   350     8   136   217   186     1  2342    27   127   17   522
```

You can use it in combination with relational operators

```r
> table(books$TOT.CHKOUT > 50)
FALSE  TRUE
3993    7
```
The `duplicated()` function will give you a logical vector of duplicated values.

```r
> z <- c("111", "222", "111", "333", "444")

> duplicated(z)
[1] FALSE FALSE TRUE FALSE FALSE

> !duplicated(z)
[1] TRUE TRUE TRUE FALSE FALSE TRUE TRUE TRUE
```
Use `duplicated()` nested in `table()` to find out how many values are dupes

```r
> table(duplicated(z))
FALSE  TRUE
   4    1
```

Use `which()` to identify the duplicated element

```r
> which(duplicated(z))
[1] 3
```
• `{dplyr}` was created by Hadley Wickham
• A "grammar for data manipulation" that simplifies a number of critical data cleaning and transformation tasks
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rename()</code></td>
<td>rename variables</td>
</tr>
<tr>
<td><code>recode()</code></td>
<td>replace data values</td>
</tr>
<tr>
<td><code>filter()</code></td>
<td>create subsets of data</td>
</tr>
<tr>
<td><code>arrange()</code></td>
<td>sort data</td>
</tr>
<tr>
<td><code>select()</code></td>
<td>choose specific variables</td>
</tr>
<tr>
<td><code>mutate()</code></td>
<td>create new variables</td>
</tr>
<tr>
<td><code>summarize()</code></td>
<td>create summaries of data</td>
</tr>
</tbody>
</table>

Hosted by ALCTS, Association for Library Collections and Technical Services
Data manipulation with dplyr

To read thorough examples of `dplyr` functions, run in your console:

```
> vignette("dplyr")
```

Make sure to load the package

```
> library(dplyr)
```
**dplyr functions**

function(data frame, arguments what to do with the data frame)

returns a data frame

rename(books, title = x245.ab)
dplyr::rename

> names(books)
[1] "CALL...BIBLIO." "X245.ab" "X245.c" "LOCATION"
"TOT.CHKOUT" "LOUTDATE"
[7] "SUBJECT" "ISN" "CALL...ITEM." "X008.Date.One"
"BCODE2" "BCODE1"
Where does \texttt{x245\_ab} come from?

\[245|ab\]

R doesn’t allow variables to start with numbers, and a period is the only valid punctuation, so R automatically converted it.

Similarly, \texttt{CALL #(ITEM)} becomes \texttt{CALL\ldots ITEM}.
rename(): Rename variables

Remember, you have to assign (<-) your function to books in order to overwrite the object

```r
> books <- rename(books
, title = x245.ab
, author = x245.c
, callnumber = CALL...BIBLIO.
, isbn = ISN
, pubyear = X008.Date.One
, subCollection = BCODE1
, format = BCODE2)
```
dplyr::recode

recode(): replace data values

First, use unique() to see all elements in a variable

```r
> unique(books$subCollection)
[1] "u" "-" "j" "r" "b" "a" "s" "c" "t" "z"
```
```r
> books$subCollection <- recode(books$subCollection,
  "-" = "general collection",
  u = "government documents",
  r = "reference",
  b = "k-12 materials",
  j = "juvenile",
  s = "special collections",
  c = "computer files",
  t = "theses",
  a = "archives",
  z = "reserves")
```
```r
> table(books$subCollection)
archives computer files general collection government documents juvenile
  14       8 2296 1049  353
K-12 materials reference reserves special collections theses
  35 157  12   49   27
```
Subsetting data frames

In the same way you use brackets [] to subset vectors, you also use them to subset data frames.

```r
> scale <- c("do", "re", "mi", "fa", "so")
> scale[1]
[1] "do"
```

However, vectors have only one direction, but dataframes have two.

```r
myDataFrame[row, column]
```
Subsetting data frames

> books[5, 2]
[1] "Fear of intimacy /"

Can also use column names:

> books[5, c("title", "TOT.CHKOUT")]
     title       TOT.CHKOUT
   5 Fear of intimacy /           7
Subsetting with brackets is good to know, but the `dplyr` package provides a much easier way with `filter()`.
Some of this data includes serials, videos, and other types of materials classified in the format column.

First, use `table()` to see how many elements are in the variable

```r
> table(books$format)
book    cd-rom database e-gov    doc    image kit/object map   map microform
3078       3      1       8       2     1  29       1   340
serial          538
```

Most of the items are books and serials, but there’s also some microforms.
dplyr::filter

filter(): Keep rows matching criteria

> booksOnly <- filter(books, format == "book")
## Relational Operators in R

<table>
<thead>
<tr>
<th>operator</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater Than</td>
</tr>
<tr>
<td>==</td>
<td>Equal To</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less Than or Equal To</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater Than or Equal To</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal To</td>
</tr>
<tr>
<td>%in%</td>
<td>Has A Match In</td>
</tr>
<tr>
<td>is.na</td>
<td>Is NA</td>
</tr>
<tr>
<td>!is.na</td>
<td>Is Not NA</td>
</tr>
</tbody>
</table>

> `help(Comparison)`
## Logical Tests in R

<table>
<thead>
<tr>
<th>operator</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>boolean AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>boolean NOT</td>
</tr>
<tr>
<td>any</td>
<td>ANY true</td>
</tr>
<tr>
<td>all</td>
<td>ALL true</td>
</tr>
</tbody>
</table>

> `help(Logic)`
dplyr::filter

filter on multiple variables:

books and serials

> books_and_serials <- filter(books, format == "book" | format == "serial")
dplyr::filter

filter on multiple variables:

books with 1 or more checkouts

> bookCheckouts <- filter(books
    , format == "book"
    , TOT.CHKOUT > 0)
How many are there?

> nrow(bookCheckouts)
[1] 3616

What percentage is that of all books?

> nrow(bookCheckouts)/nrow(booksOnly) * 100
[1] 63.93762

...not bad!
You can also use `filter()` to get rid of NA values

```
filter(books, complete.cases(callnumber))
```

or duplicate values

```
filter(books, !duplicated(title))
```

the exclamation mark means *not* duplicated
dplyr::select

**select()**: Keep (or remove) only the variables you mention

```r
> booksTitleCheckouts <- select(books, title, TOT.CHKOUT)

> dim(booksTitleCheckouts)
[1] 10000 2
```
Specify the variables you want to remove with a –

```r
> books <- select(books, -CALL...ITEM.)
```
dplyr::arrange

arrange(): Sort data by alphabetical or numerical order

> books <- arrange(books, title)

Use `desc()` to sort in descending order

> books <- arrange(books, desc(TOT.CHKOUT))
mutate(): Create new variables

Use `str_sub()` from the `stringr` package to extract the first character of the callnumber variable (the LC Class)

```r
> booksLC <- mutate(books,
  , lc.class = str_sub(callnumber, 1, 1))
```
Putting it all together with `%>%`

```r
> myBooks <- books %>%
  filter(format == "book", TOT.CHKOUT > 0) %>%
  select(title, TOT.CHKOUT) %>%
  arrange(desc(TOT.CHKOUT))
```

This does four things and is much more efficient & readable:

- Gets only books with 1 or more checkouts
- Selects only the title and checkout columns
- Arranges the data by total number of checkouts
Write.csv

> write.csv(bookCheckouts
  , "./bookCheckouts.csv
  , row.names = F)

Writes your data frame to a CSV file in the directory and file name of your choosing.
You made it (again)!

Refer to the handouts for exercises and more in-depth explanations.
Session 3: Data Analysis & Visualization

We will run some analysis and visualization

Wednesday, 5/30/2018

2:00 PM-3:00 PM (Eastern)
1:00 PM-2:00 PM (Central)
12:00 PM-1:00 PM (Mountain)
11:00 AM-12:00 PM (Pacific)

http://www.ala.org/alcts/confevents/upcoming/webinar/IntrotoR
Questions?