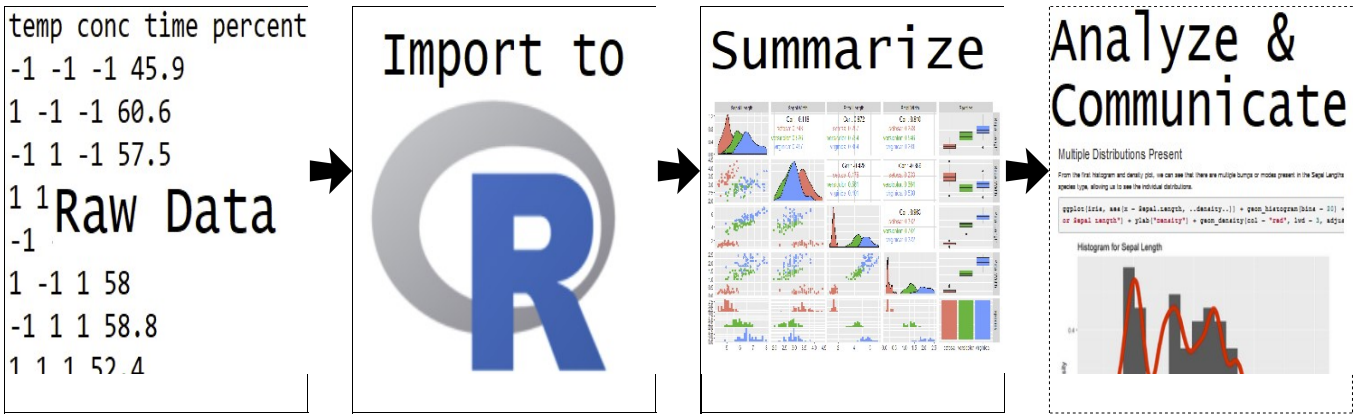
NC STATE UNIVERSITY

Introduction to R for Data Science Part II

Justin Post

What is this course about?

Basic use of R for reading, manipulating, and plotting data!



Schedule

Day 1

- Install R/R studio
- R Studio Interface
- Classes and Objects
- Attributes and Basic Data Object Manipulation
- **Reading in Data/Writing Out Data**

Reading in Data/Writing Out Data

Data comes in many formats

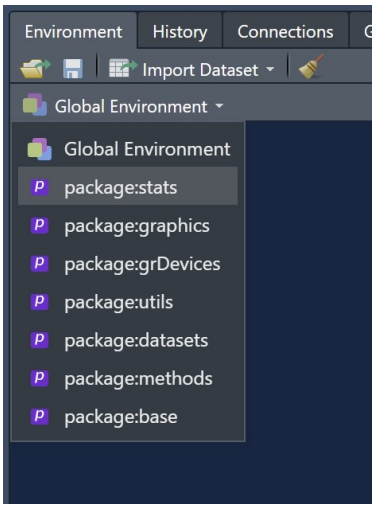
- ‘Delimited’ data: Character (such as ‘,’ , ‘>’, or [’ ’]) separated data
- Fixed field data
- Excel data
- From other statistical software, Ex: SPSS formatted data or SAS data sets
- JSON or XML data (often from a database or API)

Importing Delimited Data: Standard R Methods

- When you open R a few `packages` are loaded
- R package
  - Collection of functions/datasets/etc. in one place
  - Packages exist to do almost anything
  - List of CRAN approved packages on R's website
  - Plenty of other packages on places like GitHub

Importing Delimited Data: Standard R Methods

- When you open R a few `packages` are loaded



- `utils` package has *family* of `read.` functions ready for use!

Importing Delimited Data: Standard R Methods

Function and purpose:

Type of Delimeter	Function
Comma	`read.csv()`
Semicolon (`,` for decimal)	`read.csv2()`
Tab	`read.delim()`
White Space/General	`read.table(sep = "")`

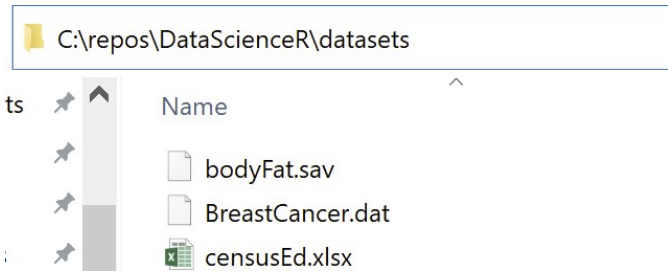
- Each function requires a **path** to the file

Reading a .csv File

- Let's read in the 'neuralgia.csv' file
- How does R locate the file?

Path to File

- Let's read in the 'neuralgia.csv' file
- How does R locate the file?
  - Can give *full path name*
    - ex: C:/repos/DataScienceR/datasets/neuralgia.csv
    - ex: C:\\repos\\DataScienceR\\datasets\\neuralgia.csv



Reading a .csv File

- Let's read in the 'neuralgia.csv' file
- Use full local path

```
neuralgiaData <- read.csv(
           "C:/repos/DataScienceR/datasets/neuralgia.csv"
           )
```

```
head(neuralgiaData)
```

```
##   Treatment Sex Age Duration Pain
## 1         P   F  68        1   No
## 2         B   M  74       16   No
## 3         P   F  67       30   No
## 4         P   M  66       26  Yes
## 5         B   F  67       28   No
## 6         B   F  77       16   No
```

Working Directory

- Let's read in the 'neuralgia.csv' file
- Using full local path not recommended!
    - Can't share code without changing path…

Working Directory

- Let's read in the 'neuralgia.csv' file
- Using full local path not recommended!
  - Can't share code without changing path…
- Can change *working directory*
  - Folder where R 'looks' for files
  - Supply **relative** path

Working Directory

- Let's read in the 'neuralgia.csv' file
- Using full local path not recommended!
  - Can't share code without changing path…
- Can change *working directory*
  - Folder where R 'looks' for files
  - Supply **relative** path

```
getwd()
```

```
## [1] "C:/repos/DataScienceR"
```

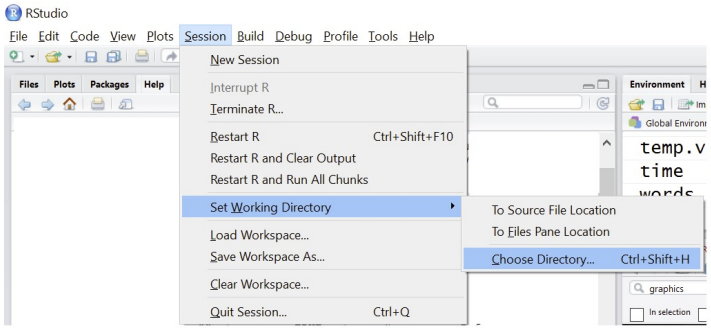
Working Directory

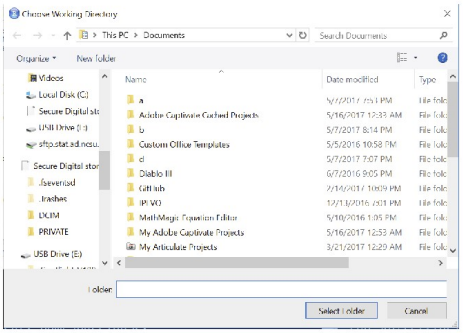
- Can change *working directory*
  - Via code

```
setwd("C:/repos/DataScienceR/datasets")
#or
setwd("C:\\repos\\DataScienceR\\datasets")
```

Working Directory

- Can change *working directory*
  - Via code
  - Via menus





Reading a .csv File

- Let's read in the 'neuralgia.csv' file

```
neuralgiaData <- read.csv("datasets/neuralgia.csv")
```

- Note: `../` drops down a folder
- As long others have the same folder structure, can share code with no path change needed!

Reading a .csv File

- Let's read in the 'neuralgia.csv' file
- R can pull from URLs as well!

```
neuralgiaData <- read.csv("https://www4.stat.ncsu.edu/~online/datasets/neuralgia.csv")
head(neuralgiaData)
```

```
##   Treatment Sex Age Duration Pain
## 1         P   F  68        1   No
## 2         B   M  74       16   No
## 3         P   F  67       30   No
## 4         P   M  66       26  Yes
## 5         B   F  67       28   No
## 6         B   F  77       16   No
```

Reading a .csv File

`read.csv()` function

Tell R where the file lives via:

- a full local path (not recommended)
- a relative path
  - can set the working directory with `setwd()`
- pulling from URL

Aside: RStudio Project

- Often have many files associated with an analysis
- With multiple analyses things get cluttered…

Aside: RStudio Project

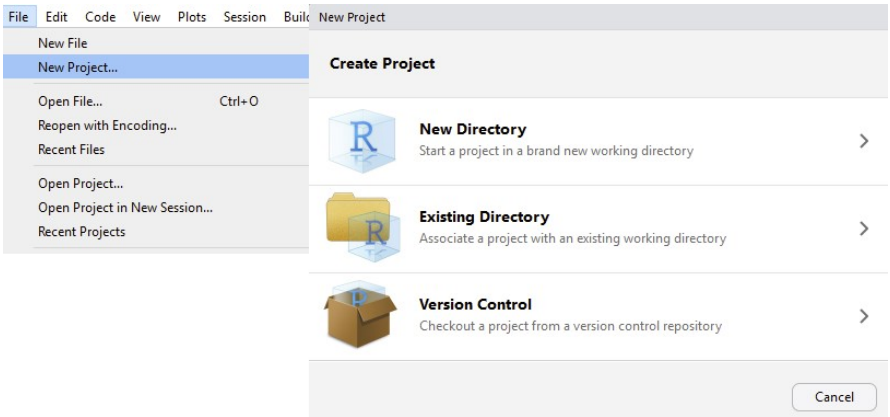
- Often have many files associated with an analysis
- With multiple analyses things get cluttered…
- Want to associate different
  - environments
  - histories
  - working directories
  - source documents

with each analysis

- Can use “Project” feature in R Studio

Aside: RStudio - Project

- Easy to create! Use an existing folder or create one:



- Easily switch between analyses!
- Create one for today's lesson
- Swap between projects using menu in top right

Reading Delimited Data

- Functions from `read.` family work well
- Concerns:
  - (formerly, prior to R 4.0) poor default function behavior
    - strings were read as `factors`

Reading Delimited Data

- Functions from `read.` family work well
- Concerns:
  - poor default function behavior
    - (formerly, prior to R 4.0) strings are read as `factors`
    - row & column names can be troublesome
  - Slower processing
  - (Slightly) different behavior on different computers

Aside: R Packages

- R package
  - Collection of functions in one place
  - Packages exist to do almost anything
  - List of CRAN approved packages on R's website
  - Plenty of other packages on places like GitHub
- "TidyVerse" - collection of R packages that share common philosophies and are designed to work together!

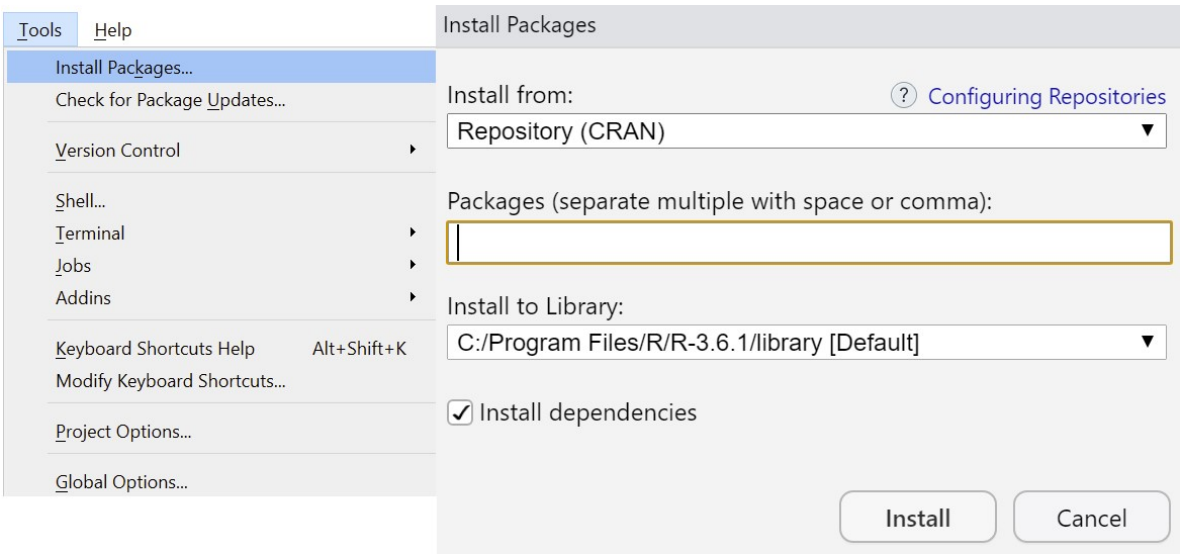
Aside: R Packages

- First time using a package
  - Must install package (download files)
  - Can use code or menus

```
install.packages("readr")
#can do multiple packages at once
install.packages(c("readr", "readxl", "haven", "DBI", "httr"))
```

Aside: R Packages

- First time using a package
  - Must install package (download files)
  - Can use code or menus



Aside: R Packages

- Only install once!
- **Each session**: read in package using `library()` or `require()`

```
library("readr")
require("haven")
```

Aside: R Packages

- Difference - if no package
  - `library()` throws an error
  - `require()` returns FALSE

```
library("notAPackage")
```

```
## Error in library("notAPackage"): there is no package called 'notAPackage'
```

```
require("notAPackage")
```

```
## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : there is no package called 'notAPackage'
```

Aside: R Packages

- Many packages to do things in R
- How to choose?
  - Want ‘fast’ code
  - Want ‘easy’ syntax
  - Good default settings on functions
  - Nice set of examples and vignettes
- Enter: TidyVerse

Aside: R Packages

- Install the `tidyverse` package

```
install.packages("tidyverse")
```

Aside: R Packages

- Install the `tidyverse` package

```
install.packages("tidyverse")
```

- Load library

```
library(tidyverse)
```

- Once library loaded, check `help(filter)`

Aside: R Packages

- Can call functions without loading full library with `::`
- If not specified, most recently loaded package takes precedent

```
dplyr::filter(neuralgiaData, Treatment == "P")
```

```
##    Treatment Sex Age Duration Pain
## 1          P   F  68        1   No
## 2          P   F  67       30   No
## 3          P   M  66       26  Yes
## 4          P   F  64        1  Yes
## 5          P   M  74        4   No
## 6          P   M  70        1  Yes
## 7          P   M  83        1  Yes
## 8          P   M  77       29  Yes
## 9          P   F  79       20  Yes
## 10         P   M  78       12  Yes
## 11         P   M  66        4  Yes
## 12         P   F  65       29   No
## 13         P   M  60       26  Yes
## 14         P   F  72       27   No
## 15         P   F  70       13  Yes
## 16         P   F  68       27  Yes
## 17         P   M  68       11  Yes
## 18         P   M  67       17  Yes
## 19         P   F  67        1  Yes
## 20         P   F  72       11  Yes
```

Aside: R Packages

Install packages first (download it)

- Can do more than one at a time

Load package with `require()` or `library()`

- Call without loading using `::`

Reading Delimited Data

We'll use the `tidyverse`!

Function and purpose:

Type of Delimeter	`utils` Function	`readr` Function
Comma	`read.csv()`	`read_csv()`
Semicolon (`,` for decimal)	`read.csv2()`	`read_csv2()`
Tab	`read.delim()`	`read_tsv()`
General	`read.table(sep = "")`	`read_delim()`
White Space	`read.table(sep = "")`	`read_table()` `read_table2()`

Reading Delimited Data

- Let's read in the 'neuralgia.csv' file

```
neuralgiaData2 <- readr::read_csv("https://www4.stat.ncsu.edu/~online/datasets/neuralgia.csv")
```

```
## Parsed with column specification:
## cols(
##   Treatment = col_character(),
##   Sex = col_character(),
##   Age = col_double(),
##   Duration = col_double(),
##   Pain = col_character()
## )
```

Reading Delimited Data

- Let's read in the 'neuralgia.csv' file

```
neuralgiaData2
```

```
## # A tibble: 60 x 5
##   Treatment Sex     Age Duration Pain
##   <chr>     <chr> <dbl>    <dbl> <chr>
## 1 P         F        68        1 No
## 2 B         M        74       16 No
## 3 P         F        67       30 No
## 4 P         M        66       26 Yes
## 5 B         F        67       28 No
## # ... with 55 more rows
```

Reading Delimited Data

- Notice: fancy printing!
- Checking column type is a basic data validation step
- `tidyverse` data frames are called `tibbles`

```
class(neuralgiaData2)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

tibbles

- Behavior slightly different than a standard `data frame`. No simplification!

```
neuralgiaData[,1]
```

```
##  [1] P B P P B B A B B A A A B A P A P A P B B A A A B P B B P P A A B B B A P B
## [39] B P P P A B A P P A B P P P B A P A P A B A
## Levels: A B P
```

```
neuralgiaData2[,1]
```

```
## # A tibble: 60 x 1
##   Treatment
##   <chr>
## 1 P
## 2 B
## 3 P
## 4 P
## 5 B
## # ... with 55 more rows
```

tibbles

- Behavior slightly different than a standard `data frame`. No simplification!
- Use either `pull()` or `$`

```
pull(neuralgiaData2, 1) #or pull(neuralgiaData2, Treatment)
```

```
##  [1] "P" "B" "P" "P" "B" "B" "A" "B" "B" "A" "A" "A" "B" "A" "P" "A" "P" "A" "P"
## [20] "B" "B" "A" "A" "A" "B" "P" "B" "B" "P" "P" "A" "A" "B" "B" "B" "A" "P" "B"
## [39] "B" "P" "P" "P" "A" "B" "A" "P" "P" "A" "B" "P" "P" "P" "B" "A" "P" "A" "P"
## [58] "A" "B" "A"
```

```
neuralgiaData2$Treatment
```

```
##  [1] "P" "B" "P" "P" "B" "B" "A" "B" "B" "A" "A" "A" "B" "A" "P" "A" "P" "A" "P"
## [20] "B" "B" "A" "A" "A" "B" "P" "B" "B" "P" "P" "A" "A" "B" "B" "B" "A" "P" "B"
## [39] "B" "P" "P" "P" "A" "B" "A" "P" "P" "A" "B" "P" "P" "P" "B" "A" "P" "A" "P"
## [58] "A" "B" "A"
```

Reading Delimited Data

- How did R determine the column types?

```
help(read_csv)
```

- Other useful inputs:
  - `skip = 0`
  - `col_names = TRUE`
  - `na = c("", "NA")`

Reading Delimited Data

- Reading *clean* delimited data pretty easy!
- Let's read in the 'chemical.txt' file (space delimited)
- `read_table2()` allows multiple white space characaters between entries

Reading Delimited Data

- Reading *clean* delimited data pretty easy!
- Let's read in the 'chemical.txt' file (space delimited)
- `read_table2()` allows multiple white space characaters between entries

```
read_table2("https://www4.stat.ncsu.edu/~online/datasets/chemical.txt")
```

```
## # A tibble: 19 x 4
##     temp  conc  time percent
##    <dbl> <dbl> <dbl>   <dbl>
##  1  -1    -1    -1      45.9
##  2   1    -1    -1      60.6
##  3  -1     1    -1      57.5
##  4   1     1    -1      58.6
##  5  -1    -1     1      53.3
##  6   1    -1     1      58
##  7  -1     1     1      58.8
##  8   1     1     1      52.4
##  9  -2     0     0      46.9
## 10   2     0     0      55.4
## 11   0    -2     0      55
## 12   0     2     0      57.5
## 13   0     0    -2      56.3
## 14   0     0     2      58.9
## 15   0     0     0      56.9
## 16   2    -3     0      61.1
## 17   2    -3     0      62.9
## 18  -1.4   2.6   0.7    60
## 19  -1.4   2.6   0.7    60.6
```

Reading Delimited Data

- Reading *clean* delimited data pretty easy!
- Let's read in the 'crabs.txt' file (tab delimited)

Reading Delimited Data

- Reading *clean* delimited data pretty easy!
- Let's read in the 'crabs.txt' file (tab delimited)

```
read_tsv("https://www4.stat.ncsu.edu/~online/datasets/crabs.txt")
```

```
## # A tibble: 173 x 6
##   color spine width satell weight     y
##   <dbl> <dbl> <dbl>  <dbl>  <dbl> <dbl>
## 1     3     3  28.3      8   3050     1
## 2     4     3  22.5      0   1550     0
## 3     2     1  26        9   2300     1
## 4     4     3  24.8      0   2100     0
## 5     4     3  26        4   2600     1
## # ... with 168 more rows
```

Reading Delimited Data

- Reading *clean* delimited data pretty easy!
- Let's read in the 'umps2012.txt' file ('>' delimited)
- Notice no column names provided
    - Year Month Day Home Away HPUmpire

Reading Delimited Data

- Reading *clean* delimited data pretty easy!
- Let's read in the 'umps2012.txt' file ('>' delimited)
- Notice no column names provided
  - Year Month Day Home Away HPUmpire

```
read_delim("https://www4.stat.ncsu.edu/~online/datasets/umps2012.txt", delim = ">",
           col_names = c("Year", "Month", "Day", "Home", "Away", "HPUmpire"))
```

```
## # A tibble: 2,359 x 6
##    Year Month   Day Home  Away  HPUmpire
##   <dbl> <dbl> <dbl> <chr> <chr> <chr>
## 1  2012     4    12 MIN   LAA   D.J. Reyburn
## 2  2012     4    12 SD    ARI   Marty Foster
## 3  2012     4    12 WSH   CIN   Mike Everitt
## 4  2012     4    12 PHI   MIA   Jeff Nelson
## 5  2012     4    12 CHC   MIL   Fieldin Culbreth
## # ... with 2,354 more rows
```

Reading Fixed Field & Tricky Non-Standard Data

- read_fwf()
  - reads in data where entries are very structured
- read_file()
  - reads an entire file into a single string
- read_lines()
  - reads a file into a character vector with one element per line
- Usually parse the last two with `regular expressions` :(

Next Up!

- Read data from other sources

Type of file	Package	Function
Delimited	`readr`	`read_csv()`, `read_tsv()`, `read_table()`, `read_delim()`
Excel (.xls,.xlsx)	`readxl`	`read_excel()`
SAS (.sas7bdat)	`haven`	`read_sas()`
SPSS (.sav)	`haven`	`read_spss()`

- Basics for JSON, databases, and APIs

Excel Data

- Read in censusEd.xlsx
- Use `read_excel()` from `readxl` package!

Excel Data

- Read in censusEd.xlsx
- Use `read_excel()` from `readxl` package!
  - Reads both xls and xlsx files
  - Detects format from extension given
  - Can't pull from web though!

Excel Data

```r
#install package if necessary
library(readxl)
#reads first sheet by default
edData <- read_excel("datasets/censusEd.xlsx")
edData
```

```
## # A tibble: 3,198 x 42
##   Area_name STCOU EDU010187F EDU010187D EDU010187N1 EDU010187N2 EDU010188F
##   <chr>     <chr>      <dbl>      <dbl> <chr>       <chr>            <dbl>
## 1 UNITED S~ 00000          0   40024299 0000        0000                 0
## 2 ALABAMA   01000          0     733735 0000        0000                 0
## 3 Autauga,~ 01001          0       6829 0000        0000                 0
## 4 Baldwin,~ 01003          0      16417 0000        0000                 0
## 5 Barbour,~ 01005          0       5071 0000        0000                 0
## # ... with 3,193 more rows, and 35 more variables: EDU010188D <dbl>,
## #   EDU010188N1 <chr>, EDU010188N2 <chr>, EDU010189F <dbl>, EDU010189D <dbl>,
## #   EDU010189N1 <chr>, EDU010189N2 <chr>, EDU010190F <dbl>, EDU010190D <dbl>,
## #   EDU010190N1 <chr>, EDU010190N2 <chr>, EDU010191F <dbl>, EDU010191D <dbl>,
## #   EDU010191N1 <chr>, EDU010191N2 <chr>, EDU010192F <dbl>, EDU010192D <dbl>,
## #   EDU010192N1 <chr>, EDU010192N2 <chr>, EDU010193F <dbl>, EDU010193D <dbl>,
## #   EDU010193N1 <chr>, EDU010193N2 <chr>, EDU010194F <dbl>, EDU010194D <dbl>,
## #   EDU010194N1 <chr>, EDU010194N2 <chr>, EDU010195F <dbl>, EDU010195D <dbl>,
## #   EDU010195N1 <chr>, EDU010195N2 <chr>, EDU010196F <dbl>, EDU010196D <dbl>,
## #   EDU010196N1 <chr>, EDU010196N2 <chr>
```

Excel Data

- Read in censusEd.xlsx
- Use `read_excel()` from `readxl` package!
  - Specify sheet with name or integers (or `NULL` for 1st) using `sheet =`
  - Can look at sheets available

```
excel_sheets("datasets/censusEd.xlsx")
```

```
##  [1] "EDU01A" "EDU01B" "EDU01C" "EDU01D" "EDU01E" "EDU01F" "EDU01G" "EDU01H"
##  [9] "EDU01I" "EDU01J"
```

```
read_excel("datasets/censusEd.xlsx", sheet = "EDU01D")
```

Excel Data

- Use `read_excel()` from `readxl` package!
  - Specify cells with contiguous range with `range =`

```
edData <- read_excel("datasets/censusEd.xlsx", sheet = "EDU01A",
                     range = cell_cols("A:D"))
edData
```

```
## # A tibble: 3,198 x 4
##   Area_name     STCOU EDU010187F EDU010187D
##   <chr>         <chr>      <dbl>      <dbl>
## 1 UNITED STATES 00000          0   40024299
## 2 ALABAMA       01000          0     733735
## 3 Autauga, AL   01001          0       6829
## 4 Baldwin, AL   01003          0      16417
## 5 Barbour, AL   01005          0       5071
## # ... with 3,193 more rows
```

Excel Data Recap

Using `read_excel()` from `readxl` package

- Reads both xls and xlsx files
- Specify sheet with name or integers (or `NULL` for 1st)
  - Use `sheet = "name"` or `sheet = #`
- Look at sheets available
  - Use `excel_sheets`
- Specify cells with continguous range
  - `range = cell_cols("...")`
  - `range = cell_rows("...")`
- Specify cells
  - `range = "R1C2:R2C5"`

SAS Data

- SAS data has extension '.sas7bdat'
- Read in smoke2003.sas7bdat
- Use `read_sas()` from `haven` package
- Not many options!

SAS Data

- SAS data has extension '.sas7bdat'
- Read in smoke2003.sas7bdat
- Use `read_sas()` from `haven` package
- Not many options!

```
#install if necessary
library(haven)
smokeData <- read_sas("https://www4.stat.ncsu.edu/~online/datasets/smoke2003.sas7bdat")
smokeData
```

```
## # A tibble: 443 x 54
##    SEQN SDDSRVYR RIDSTATR RIDEXMON RIAGENDR RIDAGEYR RIDAGEMN RIDAGEEX RIDRETH1
##   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 21010        3        2        2        2       52      633      634        3
## 2 21012        3        2        2        1       63      765      766        4
## 3 21048        3        2        1        2       42      504      504        1
## 4 21084        3        2        1        2       57      692      693        3
## 5 21093        3        2        1        2       64      778      778        2
## # ... with 438 more rows, and 45 more variables: RIDRETH2 <dbl>,
## #   DMQMILIT <dbl>, DMDBORN <dbl>, DMDCITZN <dbl>, DMDYRSUS <dbl>,
## #   DMDEDUC3 <dbl>, DMDEDUC2 <dbl>, DMDEDUC <dbl>, DMDSCHOL <dbl>,
## #   DMDMARTL <dbl>, DMDHHSIZ <dbl>, INDHHINC <dbl>, INDFMINC <dbl>,
## #   INDFMPIR <dbl>, RIDEXPRG <dbl>, DMDHRGND <dbl>, DMDHRAGE <dbl>,
## #   DMDHRBRN <dbl>, DMDHREDU <dbl>, DMDHRMAR <dbl>, DMDHSEDU <dbl>,
## #   SIALANG <dbl>, SIAPROXY <dbl>, SIAINTRP <dbl>, FIALANG <dbl>,
## #   FIAPROXY <dbl>, FIAINTRP <dbl>, MIALANG <dbl>, MIAPROXY <dbl>,
## #   MIAINTRP <dbl>, AIALANG <dbl>, WTINT2YR <dbl>, WTMEC2YR <dbl>,
## #   SDMVPSU <dbl>, SDMVSTRA <dbl>, Gender <dbl>, Age <dbl>, IncomeGroup <chr>,
## #   Ethnicity <chr>, Education <dbl>, SMD070 <dbl>, SMQ077 <dbl>, SMD650 <dbl>,
## #   PacksPerDay <dbl>, lbdvid <dbl>
```

SAS Data

- Note: Variables had SAS labels. Don't show on print!
  - Will show on `View(smokeData)` (or click on data from environment)

```
str(smokeData)
```

```
## tibble [443 x 54] (S3: tbl_df/tbl/data.frame)
##  $ SEQN       : num [1:443] 21010 21012 21048 21084 21093 ...
##   ..- attr(*, "label")= chr "Patient ID"
##  $ SDDSRVYR   : num [1:443] 3 3 3 3 3 3 3 3 3 3 ...
##   ..- attr(*, "label")= chr "Data Release Number"
##  $ RIDSTATR   : num [1:443] 2 2 2 2 2 2 2 2 2 2 ...
##   ..- attr(*, "label")= chr "Interview/Examination Status"
##  $ RIDEXMON   : num [1:443] 2 2 1 1 1 2 1 2 1 1 ...
##   ..- attr(*, "label")= chr "Six month time period"
##  $ RIAGENDR   : num [1:443] 2 1 2 2 2 2 1 2 1 2 ...
##   ..- attr(*, "label")= chr "Gender 1=M 2=F"
##  $ RIDAGEYR   : num [1:443] 52 63 42 57 64 63 66 60 65 47 ...
##   ..- attr(*, "label")= chr "Age in Years at Exam"
##  $ RIDAGEMN   : num [1:443] 633 765 504 692 778 763 801 731 786 573 ...
##   ..- attr(*, "label")= chr "Age in Months - Recode"
##  $ RIDAGEEX   : num [1:443] 634 766 504 693 778 763 801 732 787 573 ...
##   ..- attr(*, "label")= chr "Exam Age in Months - Recode"
##  $ RIDRETH1   : num [1:443] 3 4 1 3 2 3 1 3 3 3 ...
##   ..- attr(*, "label")= chr " Ethnicity 1=MexAm 2=OthHisp 3=OthCauc 4=OthBla 5=Oth"
##  $ RIDRETH2   : num [1:443] 1 2 3 1 5 1 3 1 1 1 ...
##   ..- attr(*, "label")= chr "Linked NH3 Race/Ethnicity - Recode"
##  $ DMQMILIT   : num [1:443] 2 2 2 2 2 2 2 2 1 2 ...
##   ..- attr(*, "label")= chr "Veteran/Military Status"
##  $ DMDBORN    : num [1:443] 1 1 1 1 3 1 1 1 1 1 ...
##   ..- attr(*, "label")= chr "Country of Birth - Recode"
##  $ DMDCITZN   : num [1:443] 1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "label")= chr "Citizenship Status"
##  $ DMDYRSUS   : num [1:443] NA NA NA NA 9 NA NA NA NA NA ...
##   ..- attr(*, "label")= chr "Length of time in US"
##  $ DMDEDUC3   : num [1:443] NA NA NA NA NA NA NA NA NA NA ...
##   ..- attr(*, "label")= chr "Education Level - Children/Youth 6-19"
##  $ DMDEDUC2   : num [1:443] 4 3 3 4 1 3 1 4 4 4 ...
##   ..- attr(*, "label")= chr "Education Level for Over 20"
##  $ DMDEDUC    : num [1:443] 3 2 2 3 1 2 1 3 3 3 ...
##   ..- attr(*, "label")= chr "Education - Recode (old version)"
##  $ DMDSCHOL   : num [1:443] NA NA NA NA NA NA NA NA NA NA ...
##   ..- attr(*, "label")= chr "Now attending school?"
##  $ DMDMARTL   : num [1:443] 6 6 3 1 2 1 6 3 1 1 ...
##   ..- attr(*, "label")= chr "Marital Status"
```

SAS Data

- Note: Variables had SAS labels. Don't show on print!
  - Will show on `View(smokeData)` (or click on data from environment)
  - Can access via

```
attr(smokeData$SDDSRVYR, "label")
```

```
## [1] "Data Release Number"
```

SPSS Data

- SPSS data has extension ".sav"
- Read in bodyFat.sav
- Use `read_spss()` from `haven` package
- Not many options!

SPSS Data

- SPSS data has extension ".sav"
- Read in bodyFat.sav
- Use `read_spss()` from `haven` package
- Not many options!

```
bodyFatData <- read_spss("https://www4.stat.ncsu.edu/~online/datasets/bodyFat.sav")
bodyFatData
```

```
## # A tibble: 20 x 4
##        y    x1    x2    x3
##    <dbl> <dbl> <dbl> <dbl>
##  1  19.5  43.1  29.1  11.9
##  2  24.7  49.8  28.2  22.8
##  3  30.7  51.9  37    18.7
##  4  29.8  54.3  31.1  20.1
##  5  19.1  42.2  30.9  12.9
##  6  25.6  53.9  23.7  21.7
##  7  31.4  58.5  27.6  27.1
##  8  27.9  52.1  30.6  25.4
##  9  22.1  49.9  23.2  21.3
## 10  25.5  53.5  24.8  19.3
## 11  31.1  56.6  30    25.4
## 12  30.4  56.7  28.3  27.2
## 13  18.7  46.5  23    11.7
## 14  19.7  44.2  28.6  17.8
## 15  14.6  42.7  21.3  12.8
## 16  29.5  54.4  30.1  23.9
## 17  27.7  55.3  25.7  22.6
## 18  30.2  58.6  24.6  25.4
## 19  22.7  48.2  27.1  14.8
## 20  25.2  51    27.5  21.1
```

Recap

- Reading Data

Type of file	Package	Function
Delimited	`readr`	`read_csv()`, `read_tsv()`, `read_table()`, `read_delim()`
Excel (.xls,.xlsx)	`readxl`	`read_excel()`
SAS (.sas7bdat)	`haven`	`read_sas()`
SPSS (.sav)	`haven`	`read_spss()`

Writing Out Data

Writing Data

- Usually write to .csv (or other delimiter)
- Use `write_csv()` from `readr` package
- Check help!
    - Will write to path or working directory

```
write_csv(x = smokeData,
          path = "C:/repos/DataScienceR/datasets/smokeData.csv")
```

Reading in Data/Writing Out Data

Writing Data

- Usually write to .csv (or other delimiter)
- Use `write_csv()` from `readr` package
- Check help!
  - Will write to path or working directory
  - `append` option won't overwrite but structures must match…

```
write_csv(x = bodyFatData,
          path = "C:/repos/DataScienceR/datasets/smokeData.csv",
          append = TRUE)
```

Resources for Other Data Sources

JSON - JavaScript Object Notation

- Used widely across the internet and databases
- Can represent usual 2D data or heirarchical data

Resources for Other Data Sources

JSON - JavaScript Object Notation

- Uses key-value pairs

```
{
  {
    "name": "Barry Sanders"
    "games" : 153
    "position": "RB"
  },
  {
    "name": "Joe Montana"
    "games": 192
    "position": "QB"
  }
}
```

Resources for Other Data Sources

JSON - JavaScript Object Notation

Three major R packages

1. `rjson`
2. `RJSONIO`
3. `jsonlite`
   - many nice features
   - a little slower implementation

Resources for Other Data Sources

JSON - JavaScript Object Notation

`jsonlite` basic functions:

Function	Description
`fromJSON`	Reads JSON data from file path or character string. Converts and simplfies to R object
`toJSON`	Writes R object to JSON object
`stream_in`	Accepts a *file connection* - can read streaming JSON data

Resources for Other Data Sources

APIs - Application Programming Interfaces

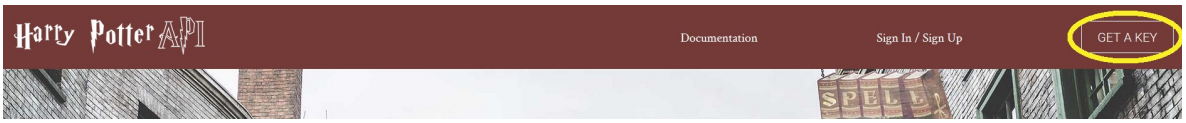
A defined method for asking for information from a computer

- Useful for getting data
- Useful for allowing others to run your model without a GUI (like Shiny)
- Many open APIs, just need key
- Often just need to construct proper URL

Resources for Other Data Sources

APIs - Quick Example

- Query Harry Potter database https://www.potterapi.com/
- Get key in top right (sign up for account)



Resources for Other Data Sources

APIs - Quick Example

- Query Harry Potter database https://www.potterapi.com/
- Documentation:
  - All routes need to be prefixed with https://www.potterapi.com/v1/
  - GET request: /spells returns all spells
  - Key goes on the end

```
baseURL <- "https://www.potterapi.com/v1/"
value <- "spells?"
key <- "key=$2a$10$UMvDCH.93fa2KOjKbJYkOOPMNzdzQpJ0gMnVEtcHzW5Ic04HUmcsa"
URL <- paste0(baseURL, value, key)
#spellData <- RCurl::getURL(URL) #Website currently down...
```

Resources for Other Data Sources

APIs - Quick Example

- Query Harry Potter database https://www.potterapi.com/
- Default response format is JSON

Resources for Other Data Sources

APIs - Quick Example

- Query Harry Potter database https://www.potterapi.com/
- Default response format is JSON

```
spellDataDF <- jsonlite::fromJSON(spellData)
as_tibble(spellDataDF)
```

Resources for Other Data Sources

APIs - Application Programming Interfaces

Access in R

- Article here discusses accessing APIs generically with R
- Same website gives a list of APIs

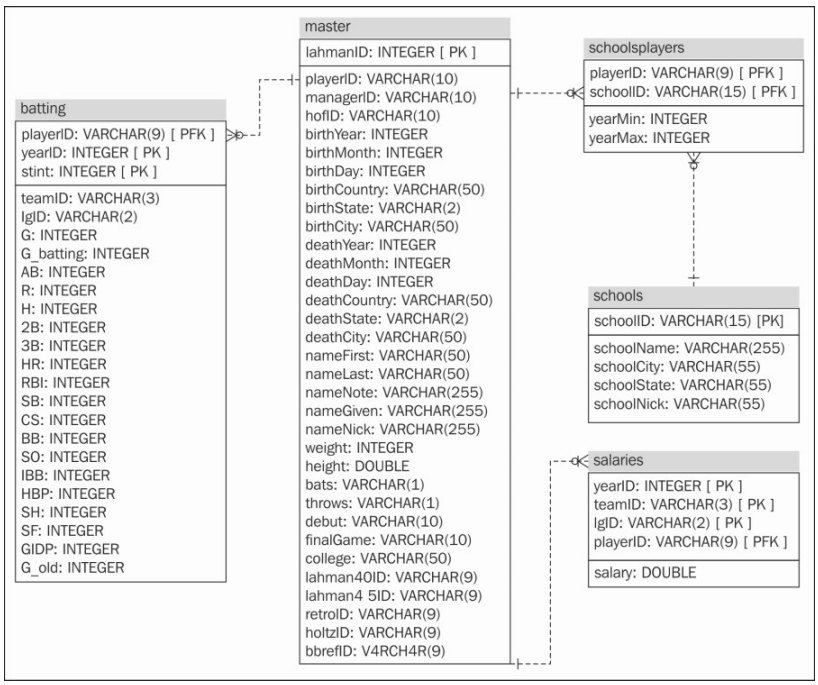
Resources for Other Data Sources

Databases

- Collection of data, usually a bunch of (related) 2D tables

Resources for Other Data Sources

Example database structure



Source: oreilly.com

Resources for Other Data Sources

Databases

- Collection of data, usually a bunch of (related) 2D tables
- Relational Database Management System (RDBMS) controls how users interact
- Structured Query Language (SQL) - language used by RDBMS

Resources for Other Data Sources

Databases

- Collection of data, usually a bunch of 2D tables
- Relational Database Management System (RDBMS) controls how users interact
- Structured Query Language (SQL) - language used by RDBMS
  - Used to obtain data from tables
  - Used to combine data from separate tables (‘keys’ relate tables)
  - Used to manipulate and create variables, structure and edit databases, etc.

Resources for Other Data Sources

Databases

Many popular RDBMS, some free some proprietary (often referred to as databases…)

- Oracle - most popular (cross platform)
- SQL Server - Microsoft product
- DB2 - IBM product
- MySQL (open source) - Not as many features but popular
- PostgreSQL (open source)

Basic SQL language constant across all - features differ

Resources for Other Data Sources

Databases - Common flow in R

1. Connect to the database with `DBI::dbConnect()`

- Need appropriate R package for database backend
  - `RSQLite::SQLite()` for RSQLite
  - `RMySQL::MySQL()` for RMySQL
  - `RPostgreSQL::PostgreSQL()` for RPostgreSQL
  - `odbc::odbc()` for Open Database Connectivity
  - `bigrquery::bigquery()` for google's bigQuery

```
con <- DBI::dbConnect(RMySQL::MySQL(),
  host = "hostname.website",
  user = "username",
  password = rstudioapi::askForPassword("DB password")
)
```

Resources for Other Data Sources

Databases - Common flow in R

1. Connect to the database with `DBI::dbConnect()`

- Need appropriate R package for database backend

2. Use `tbl()` to reference a table in the database

```
tbl(con, "name_of_table")
```

Resources for Other Data Sources

Databases - Common flow in R

1. Connect to the database with `DBI::dbConnect()`

- Need appropriate R package for database backend

2. Use `tbl()` to reference a table in the database
3. Query the database with `SQL` or `dplyr/dbplyr` (we'll learn `dplyr` soon!)

Resources for Other Data Sources

Databases - Common flow in R

1. Connect to the database with `DBI::dbConnect()`

- Need appropriate R package for database backend

2. Use `tbl()` to reference a table in the database
3. Query the database with `SQL` or `dplyr/dbplyr` (we'll learn `dplyr` soon!)
4. Disconnect from database with `dbDisconnect()`

Resources for Other Data Sources

Databases - Quick Example

- Connect to Google's BigQuery database

```
#devtools::install_github("r-dbi/bigrquery")
library(DBI)
con <- dbConnect(
  bigrquery::bigquery(),
  project = "publicdata",
  dataset = "samples",
  billing = "your-project-id-here"
  )
```

Resources for Other Data Sources

Databases - Quick Example

- Connect to Google's BigQuery database

```
dbListTables(con)
natality <- tbl(con, "natality")

natality %>%
  select(starts_with("mother"), year, cigarette_use, weight_pounds) %>%
  collect()

dbDisconnect(con)
```

- More about R Studio and Databases

Recap

- Read data from other sources

Type of file	Package	Function
Delimited	`readr`	`read_csv()`,`read_tsv()`,`read_table()`,`read_delim()`
Excel (.xls,.xlsx)	`readxl`	`read_excel()`
SAS (.sas7bdat)	`haven`	`read_sas()`
SPSS (.sav)	`haven`	`read_spss()`

- Resources for JSON, databases, and APIs

Activity

- **Reading/Writing Data Activity** instructions available on web
- We'll send you to breakout rooms
- One TA or instructor in each room to help out
- Feel free to ask questions about anything you didn't understand as well!