Introduction to R (Part I)

Justin Post

March 20, 2018

What do we want to be able to do?

- Read in data
- Manipulate data
- Plot data
- Summarize data
- Analyze data

Why learn R?

- It's free, open source, available on all major platforms.

Why learn R?

- It's free, open source, available on all major platforms.
- Access to the newest methods, *easy* data manipulation, and plotting.

Why learn R?

- It's free, open source, available on all major platforms.
- Access to the newest methods, *easy* data manipulation, and plotting.
- Can automate analysis and data manipulation more easily than programs like Excel.

Why learn R?

- It's free, open source, available on all major platforms.
- Access to the newest methods, *easy* data manipulation, and plotting.
- Can automate analysis and data manipulation more easily than programs like Excel.
- Great community support (stackoverflow, R-help mailing list, etc.)

Why learn R?

- It's free, open source, available on all major platforms.
- Access to the newest methods, *easy* data manipulation, and plotting.
- Can automate analysis and data manipulation more easily than programs like Excel.
- Great community support (stackoverflow, R-help mailing list, etc.)
- Ability to create pdfs, slides, reports, html files, and interactive apps.

Example Analysis

Manipulating and analyzing census data

	A	B	C	D	E	F	G	H	
1	Area_name	STCOU	EDU010187F	EDU010187D	EDU010187N1	EDU010187N2	EDU010188F	EDU010188D	ED
2	UNITED STATES	00000	0	40024299	0000	0000	0	39967624	00
3	ALABAMA	01000	0	733735	0000	0000	0	728234	00
4	Autauga, AL	01001	0	6829	0000	0000	0	6900	00
5	Baldwin, AL	01003	0	16417	0000	0000	0	16465	00
6	Barbour, AL	01005	0	5071	0000	0000	0	5098	00
7	Bibb, AL	01007	0	3557	0000	0000	0	3508	00

- Columns that end with `D` are public school enrollment
- Across multiple sheets
- Want to plot data for states
- Data format common across many data sets!

Read in/Manipulate Data

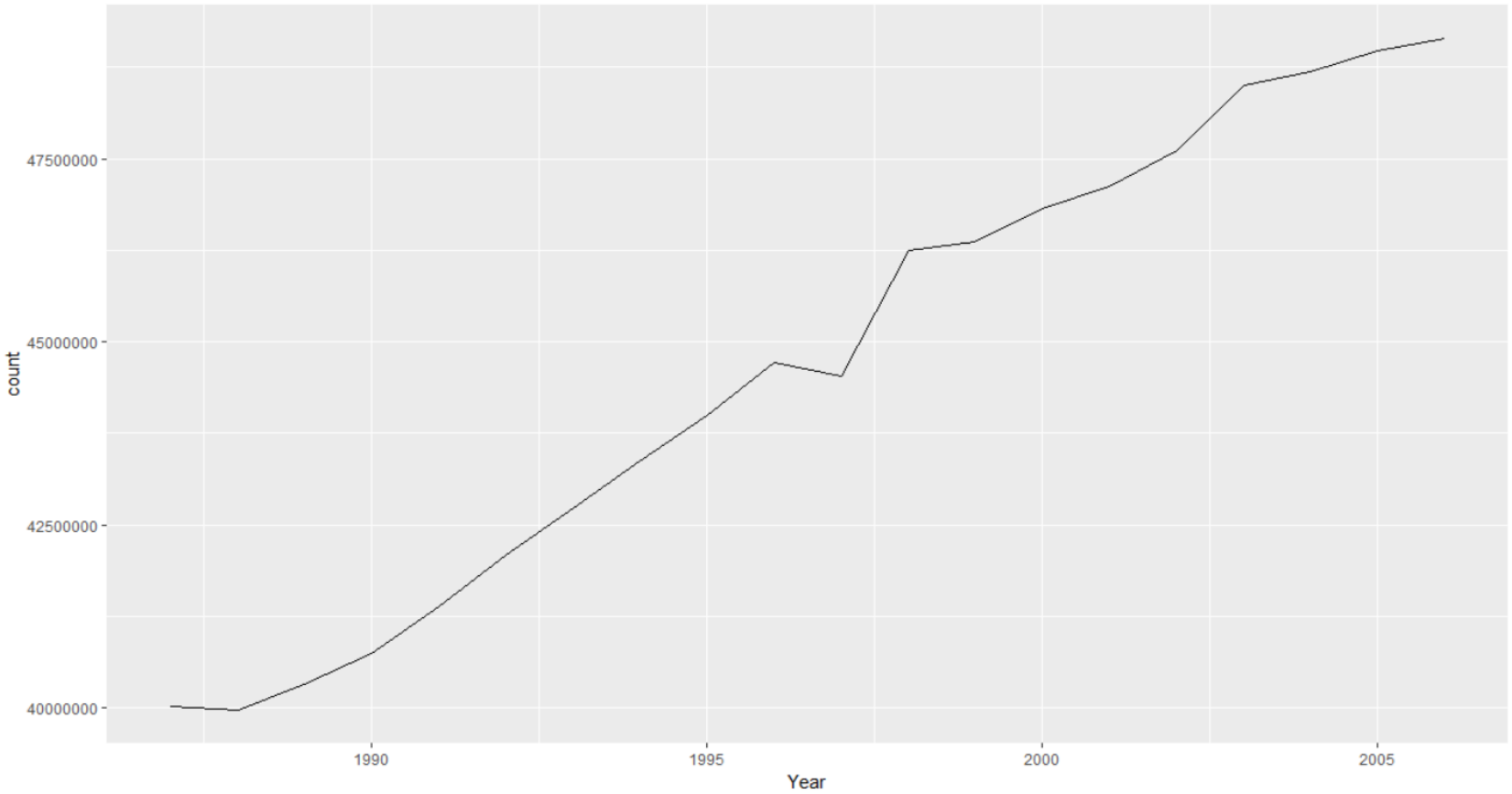
```
#read in census data
temp1 <- read_excel("datasets\\EDU01.xls", sheet = 1)
temp2 <- read_excel("datasets\\EDU01.xls", sheet = 2)

#combine
temp <- cbind(temp1, temp2[,-c(1:2)])
temp <- select(temp, Area_name, STCOU, ends_with("D"))
full <- gather(temp, key = Year, value = count,
                                ends_with("D"))
full$Year <- grabYear(full$Year)
```

Modified Data

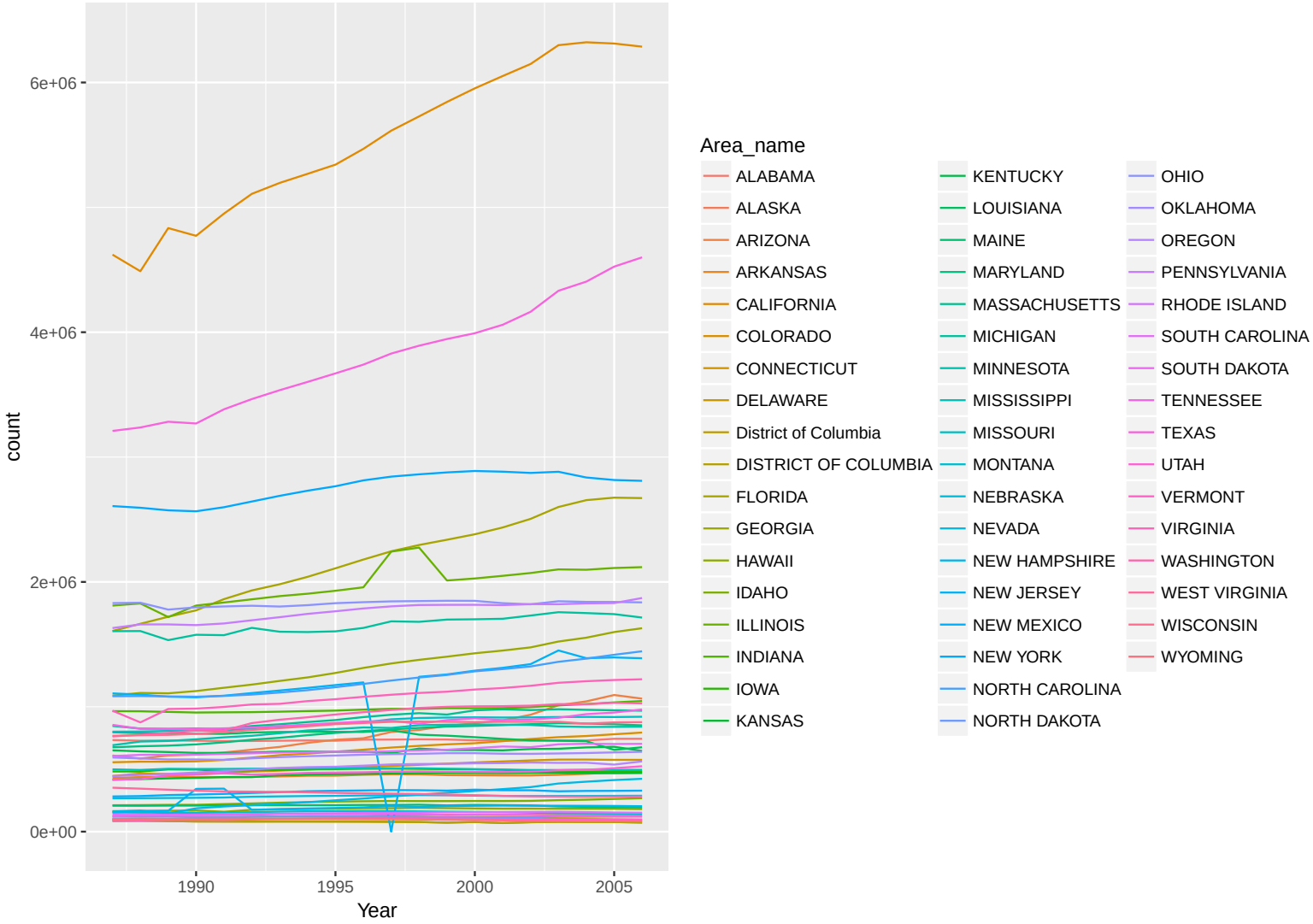
```
## # A tibble: 63,960 x 4
##       Area_name STCOU  Year    count
##           <chr> <chr> <dbl>    <dbl>
## 1 UNITED STATES 00000  1987 40024299
## 2       ALABAMA 01000  1987   733735
## 3   Autauga, AL 01001  1987     6829
## 4   Baldwin, AL 01003  1987    16417
## 5   Barbour, AL 01005  1987     5071
## # ... with 6.396e+04 more rows
```

Plot data

```
ggplot(filter(full, Area_name == "UNITED STATES"),
              aes(x= Year, y = count))+geom_line()
```



Plot data


6e+06
4e+06
2e+06
0e+00
count
1990
1995
2000
2005
Year
Area_name
ALABAMA
ALASKA
ARIZONA
ARKANSAS
CALIFORNIA
COLORADO
CONNECTICUT
DELAWARE
District of Columbia
DISTRICT OF COLUMBIA
FLORIDA
GEORGIA
HAWAII
IDAHO
ILLINOIS
INDIANA
IOWA
KANSAS
KENTUCKY
LOUISIANA
MAINE
MARYLAND
MASSACHUSETTS
MICHIGAN
MINNESOTA
MISSISSIPPI
MISSOURI
MONTANA
NEBRASKA
NEVADA
NEW HAMPSHIRE
NEW JERSEY
NEW MEXICO
NEW YORK
NORTH CAROLINA
NORTH DAKOTA
OHIO
OKLAHOMA
OREGON
PENNSYLVANIA
RHODE ISLAND
SOUTH CAROLINA
SOUTH DAKOTA
TENNESSEE
TEXAS
UTAH
VERMONT
VIRGINIA
WASHINGTON
WEST VIRGINIA
WISCONSIN
WYOMING

Drawbacks of Using R

- Slow for large problems (can be sped up!)

Drawbacks of Using R

- Slow for large problems (can be sped up!)
- Code style differs greatly across R!

Drawbacks of Using R

- Slow for large problems (can be sped up!)
- Code style differs greatly across R!
- New code not necessarily verified

Drawbacks of Using R

- Slow for large problems (can be sped up!)
- Code style differs greatly across R!
- New code not necessarily verified
- Often many ways to do the same thing

Where do we start?

- Install R/R studio/R Studio Interface
- Common Data Objects
- Reading in Data/Writing Out Data
- Subsetting/Manipulating Data
- Summaries of Data
- Basic Analyses

Installing R

- Info on installing R and R studio available here

- Let's take a few minutes and make sure everyone has these installed and working properly!

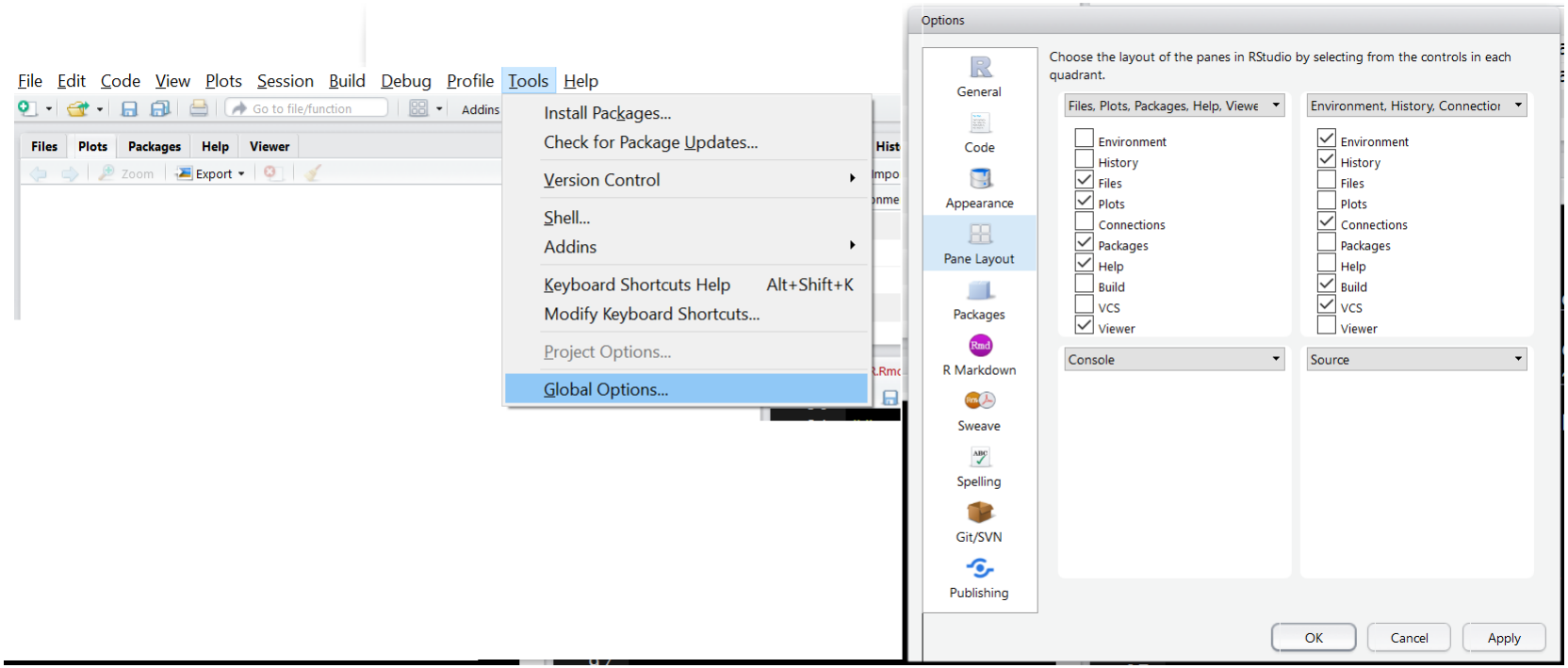
R Studio Interface

Four main 'areas' we'll use

- Scripting and Viewing Area
- Workspace/History
- Plots/Help
- Console

R Studio Interface

To rearrange panes



- Global options –> Appearance allows font/background changes
- Global options –> Code allows for soft wrap of code

Basic Use of R

- You can type directly into the console to **evaluate** code
- R is the fanciest calculator you could ever want!

```
#simple math operations (# is a comment, not evaluated)
3 + 7
```

```
## [1] 10
```

```
10 * exp(3)
```

```
## [1] 200.8554
```

```
log(pi^2) #log is natural log by default
```

```
## [1] 2.28946
```

Basic Use or R

- Usually want to keep code for later use

  - Write code in a 'script'

Basic Use or R

- `Usually want to keep code for later use`

  - Write code in a 'script'
  - Save code script

Basic Use or R

- `Usually want to keep code for later use`

  - Write code in a 'script'
  - Save code script
  - Send lines from script to console via:
    "Run" button (runs current line)
    CTRL+r (PC) or Command+Enter (MAC)
    Highlight section and do above

Objects and Common Classes

- Often want to save result for later use
- Can store output in an R 'object'

```
#save for later
avg <- (5 + 7 + 6) / 3
#call avg object
avg
```

```
## [1] 6
```

```
#strings can be saved as well
words <- "Hello there!"
words
```

```
## [1] "Hello there!"
```

Objects and Common Classes

- Objects are fundamental in R
- R objects store *results* of code
- Calling R object then prints results. No repeat computations!
- Many *classes* of objects, we'll look at a few!

Objects and Common Classes

- Five major data structures used
  1. Atomic Vector (1d)
  2. Matrix (2d)
  3. Array (nd) (we'll skip)
  4. Data Frame (2d)
  5. List (1d)

Objects and Common Classes

1. Atomic Vector (a set of elements with an ordering)
   - c() function "combines" values together

```
#vectors (1 dimensional) objects
#all elements of the same 'type'
x <- c(1, 3, -20, sqrt(2))
y <- c("cat", "dog", "bird", "floor")
x
```

```
## [1]   1.000000   3.000000 -20.000000   1.414214
```

```
y
```

```
## [1] "cat"   "dog"   "bird"  "floor"
```

Objects and Common Classes

- Many ways to populate a numeric vector

```
1:5 / 20
```

```
## [1] 0.05 0.10 0.15 0.20 0.25
```

```
seq(from = 1, to = 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```
runif(4, min = 0, max = 1)
```

```
## [1] 0.66978296 0.52015754 0.41350921 0.09724422
```

Help Files

- Functions are ubiquitous in R!
- To find out about a function's arguments use `help()`

Help Files

- Functions are ubiquitous in R!
- To find out about a function's arguments use `help()`
- Understanding the help files is key to using code!

Help Files

- Functions are ubiquitous in R!
- To find out about a function's arguments use `help()`
- Understanding the help files is key to using code!
- For instance we can try:
  `help(seq)`
  `help(runif)`

Objects and Common Classes

2. Matrix (collection of vectors of the same **type and length**)

```
#populate vectors
x <- rep(0.2, times = 6)
y <- c(1, 3, 4, -1, 5, 6)

#check 'type'
is.numeric(x)
```

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

```
is.numeric(y)
```

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

Objects and Common Classes

2. Matrix (collection of vectors of the same **type and length**)

```
#populate vectors
x <- rep(0.2, times = 6)
y <- c(1, 3, 4, -1, 5, 6)

#check 'length'
length(x)
```

```
## [1] 6
```

```
length(y)
```

```
## [1] 6
```

Objects and Common Classes

2. Matrix (collection of vectors of the same **type and length**)

- Create a matrix with `matrix()`
- Check `help(matrix)`

Objects and Common Classes

2. Matrix (collection of vectors of the same **type and length**)

```
#populate vectors
x <- rep(0.2, times = 6)
y <- c(1, 3, 4, -1, 5, 6)
#combine in a matrix (check help(matrix))
matrix(c(x, y), ncol = 2)
```

```
##      [,1] [,2]
## [1,]  0.2    1
## [2,]  0.2    3
## [3,]  0.2    4
## [4,]  0.2   -1
## [5,]  0.2    5
## [6,]  0.2    6
```

Objects and Common Classes

2. Matrix (collection of vectors of the same **type and length**)

```
x <- c("Hi", "There", "!")
y <- c("a", "b", "c")
z <- c("One", "Two", "Three")
matrix(c(x, y, z), nrow = 3)
```

```
##      [,1]    [,2] [,3]
## [1,] "Hi"    "a"  "One"
## [2,] "There" "b"  "Two"
## [3,] "!"     "c"  "Three"
```

Objects and Common Classes

4. Data Frame (collection (list) of vectors of the same **length**)

```
x <- c("a", "b", "c", "d", "e", "f")
y <- c(1, 3, 4, -1, 5, 6)
z <- 10:15
data.frame(x, y, z)
```

```
##   x  y  z
## 1 a  1 10
## 2 b  3 11
## 3 c  4 12
## 4 d -1 13
## 5 e  5 14
## 6 f  6 15
```

Objects and Common Classes

4. Data Frame (collection (list) of vectors of the same **length**)

```
x <- c("a", "b", "c", "d", "e", "f")
y <- c(1, 3, 4, -1, 5, 6)
z <- 10:15
data.frame(char = x, data1 = y, data2 = z)
```

- char, data1, and data2 become the variable names for the data frame
- Most used object type for data!

Objects and Common Classes

5. List (vector that can have differing **elements**)

```
list("Hi", 1, 2, "!")
```

```
## [[1]]
## [1] "Hi"
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] 2
##
## [[4]]
## [1] "!"
```

Objects and Common Classes

5. List (vector that can have differing **elements**)
   - Not just differing types, but differing objects!

```
x <- c("Hi", "There", "!")
y <- c(1, 3, 4, -1, 5, 6)
list(x, y)
```

```
## [[1]]
## [1] "Hi"    "There" "!"
##
## [[2]]
## [1]  1  3  4 -1  5  6
```

   - More flexible than a Data Frame!

Recap!

Review:

Dimension	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data Frame

- For most data analysis you'll use data frames!
- Next up: How do we access/change parts of our objects?

Activity

- **Objects and Common Classes Activity** instructions available on web
- Feel free to work in small groups
- Feel free to ask questions about anything you didn't understand or the activity!

Basic Data Manipulation

- **How do we access different parts of our object?**

Basic Data Manipulation

- **How do we access different parts of our object?**
- Often want things like
  - Just a column

Basic Data Manipulation

- **How do we access different parts of our object?**
- Often want things like
  - Just a column
  - Multiple columns

Basic Data Manipulation

- **How do we access different parts of our object?**
- Often want things like
  - Just a column
  - Multiple columns
  - Just a row

Basic Data Manipulation

- **How do we access different parts of our object?**
- Often want things like
  - Just a column
  - Multiple columns
  - Just a row
  - Multiple rows

Basic Data Manipulation

- **How do we access different parts of our object?**
- Often want things like
  - Just a column
  - Multiple columns
  - Just a row
  - Multiple rows

- Let's go through each of our common data types and work our way up!

Basic Data Manipulation

Atomic Vectors

- Access elements of a vector using square brackets

```
letters #built in vector
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "r
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
letters[10]
```

```
## [1] "j"
```

Basic Data Manipulation

Atomic Vectors

- Can feed R a vector of values to choose

```
letters[1:4]
```

```
## [1] "a" "b" "c" "d"
```

```
letters[c(5, 10, 15, 20, 25)]
```

```
## [1] "e" "j" "o" "t" "y"
```

```
x <- c(1, 2, 5); letters[x]
```

```
## [1] "a" "b" "e"
```

Basic Data Manipulation

Matrices

- Access elements of a matrix using square brackets with a comma
- Notice the default row names and column names!

```
mat <- matrix(c(1:4, 20:17), ncol = 2)
mat
```

```
##      [,1] [,2]
## [1,]    1   20
## [2,]    2   19
## [3,]    3   18
## [4,]    4   17
```

Basic Data Manipulation

Matrices

- Access elements using square brackets with a comma

```
mat[2, 2]
```

```
## [1] 19
```

```
mat[ , 1]
```

```
## [1] 1 2 3 4
```

```
mat[2, ]
```

```
## [1]  2 19
```

Basic Data Manipulation

Matrices

- Access elements using square brackets with a comma

```
mat[2:4, 1]
```

```
## [1] 2 3 4
```

```
mat[c(2, 4), ]
```

```
##      [,1] [,2]
## [1,]    2   19
## [2,]    4   17
```

Basic Data Manipulation

Matrices

- Can give columns names and use them for access
- `help(matrix)` can show us how!

Basic Data Manipulation

Matrices

- Can give columns names and use them for access

```
mat <- matrix(c(1:4, 20:17), ncol = 2,
           dimnames = list(NULL,
                  c("First", "Second"))
         )
mat
```

```
##      First Second
## [1,]     1     20
## [2,]     2     19
## [3,]     3     18
## [4,]     4     17
```

Basic Data Manipulation

Matrices

- Can give columns names and use them for access

```
mat[, "First"]
```

```
## [1] 1 2 3 4
```

Basic Data Manipulation

Data Frames

- Built in iris data frame

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 .
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",.
```

Basic Data Manipulation

Data Frames

- Can access just like a matrix

```
iris[1:4, 2:4]
```

```
##   Sepal.Width Petal.Length Petal.Width
## 1         3.5          1.4         0.2
## 2         3.0          1.4         0.2
## 3         3.2          1.3         0.2
## 4         3.1          1.5         0.2
```

```
iris[1, ]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
```

Basic Data Manipulation

Data Frames

- Can use variable names

```
iris[ , c("Sepal.Length", "Species")]
```

```
##     Sepal.Length    Species
## 1            5.1     setosa
## 2            4.9     setosa
## 3            4.7     setosa
## 4            4.6     setosa
## 5            5.0     setosa
## 6            5.4     setosa
## 7            4.6     setosa
## 8            5.0     setosa
## 9            4.4     setosa
```

Basic Data Manipulation

Data Frames

- Dollar sign most common way to access columns!

```
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4
##  [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4
##  [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4
##  [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5
##  [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5
##  [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5
## [103] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6
## [120] 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6
## [137] 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5
```

Basic Data Manipulation

Lists

- Use double square brackets to get at list elements

```
x <- list("HI", c(10:20), 1)
x[[1]]
```

```
## [1] "HI"
```

```
x[[2]][4:5]
```

```
## [1] 13 14
```

Basic Data Manipulation

Lists

- If named list elements, can use $

```
x <- list("HI", Second =c(10:20), 1)
str(x)
```

```
## List of 3
##  $        : chr "HI"
##  $ Second: int [1:11] 10 11 12 13 14 15 16 17 18 19 ...
##  $        : num 1
```

```
x$Second
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

Basic Data Manipulation

Data Frames (Really a list of equal length vectors!)

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 .
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",.
```

```
iris[[2]]
```

```
##   [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3
##  [18] 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3
##  [35] 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3
```

Recap!

Accessing common data structures

- Atomic vectors - `x[ ]`
- Matrices - `x[ , ]`
- Data Frames - `x[ , ]` or `x$name`
- Lists - `x[[ ]]` or `x$name`

Activity

- **Basic Data Manipulation Activity** instructions available on web
- Feel free to work in small groups
- Feel free to ask questions about anything you didn't understand or the activity!

What do we want to be able to do?

- **Read in data**
- **Manipulate data**
- Plot data
- Summarize data
- Analyze data

Reading in Data/Writing Out Data

Data comes in many formats

- 'Delimited' data: Character (such as ',' , '>', or [' ']) separated data

Reading in Data/Writing Out Data

Data comes in many formats

- 'Delimited' data: Character (such as ',' , '>', or [' ']) separated data
- Excel data

Reading in Data/Writing Out Data

Data comes in many formats

- 'Delimited' data: Character (such as ',' , '>', or [' ']) separated data
- Excel data
- Many others!

Reading in Data/Writing Out Data

Data comes in many formats

- 'Delimited' data: Character (such as ',' , '>', or [' ']) separated data
- Excel data
- Many others!
- Many ways to read in the data... How to choose?

Reading in Data/Writing Out Data

Possible methods to read data

- Base R (what comes installed)
- Use an R 'package'

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

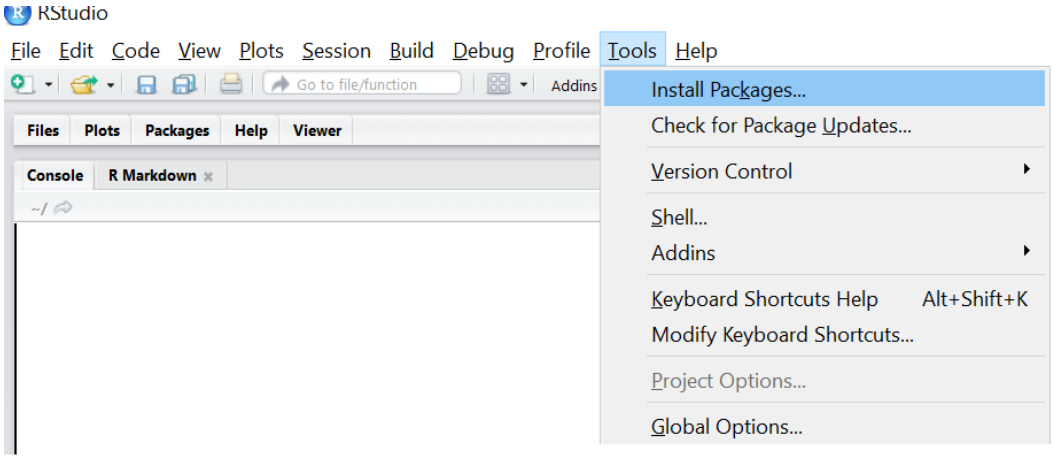
Reading in Data/Writing Out Data

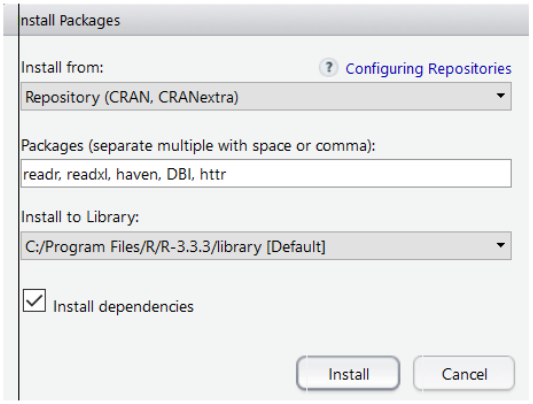
- First time using a package
  - Must install package (download)
  - Can use code or menus

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

Reading in Data/Writing Out Data

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





Reading in Data/Writing Out Data

- Once 'installed' on computer, never need to install again (unless you update R)
- **Each session** read the package in using `library()` or `require()`

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

```
## Loading required package: haven
```

Reading in Data/Writing Out Data

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

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

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

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

```
## Loading required package: notAPackage
```

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

Reading in Data/Writing Out Data

- Many packages to read in data

- How to choose?
  Want 'fast' code
  Want 'easy' syntax
  Good default settings on functions

Reading in Data/Writing Out Data

- Many packages to read in data
- How to choose?
  Want 'fast' code
  Want 'easy' syntax
  Good default settings on functions
- Base R has reasonable defaults and syntax but functions are slow

Reading in Data/Writing Out Data

- Many packages to read in data
- How to choose?
  Want 'fast' code
  Want 'easy' syntax
  Good default settings on functions
- Base R has reasonable defaults and syntax but functions are slow
- "TidyVerse" - collection of R packages that are fast, share common philosophies, and are designed to work together!

Reading in Data/Writing Out Data

Reading in a comma separated value (.csv) file

- Let's install the `tidyverse` package

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

Reading in Data/Writing Out Data

Reading in a comma separated value (.csv) file

- Let's install the `tidyverse` package

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

- Load library

```
library(tidyverse)
```

- Once library loaded, check `help(read_csv)`
- Want to read in scores.csv file using `read_csv()`

Reading in Data/Writing Out Data

- How does R locate the file?

Reading in Data/Writing Out Data

- How does R locate the file?
  - Can give file *full path name*
    - ex: E:/Other/DataScienceR/datasets/data.txt

Reading in Data/Writing Out Data

- How does R locate the file?
  - Can give file *full path name*
    - ex: E:/Other/DataScienceR/datasets/data.txt
  - Can change *working directory*
    - Folder on computer usually
    - Where R 'looks' for files
    - Supply abbreviated path name

```
getwd()
```

```
## [1] "E:/Other/DataWorks"
```

Reading in Data/Writing Out Data

- How does R locate the file?
  - Can change *working directory*

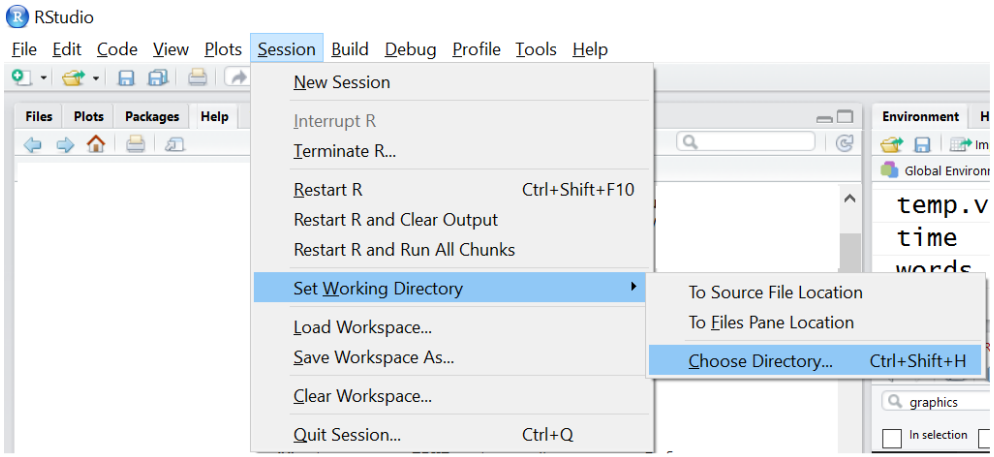
Reading in Data/Writing Out Data

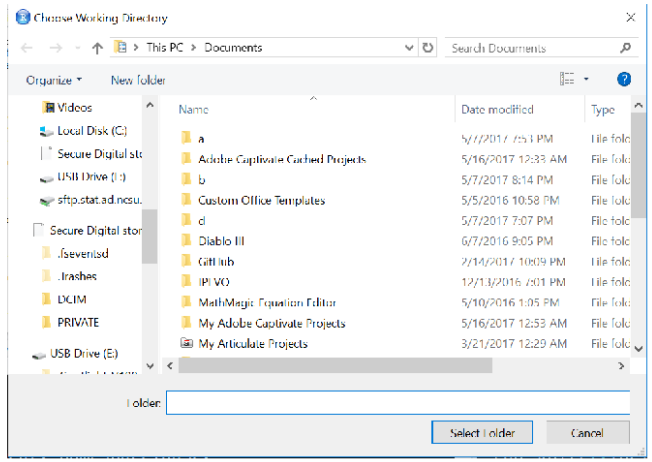
- How does R locate the file?
  - Can change *working directory*
  - Via code

```
setwd("E:\\Other\\DataWorks")
#or
setwd("E:/Other/DataWorks")
```

Reading in Data/Writing Out Data

- How does R locate the file?
  - Can change *working directory*
  - Via menus





Reading in Data/Writing Out Data

Reading in a comma separated value (.csv) file

- Often, create a folder with all files for your project
- Set working directory to that folder
- Read in data

Reading in Data/Writing Out Data

Reading in a comma separated value (.csv) file

- Checking column types a basic data validation step

```
scoreData <- read_csv(file = "datasets/scores.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   week = col_character(),
##   date = col_character(),
##   day = col_character(),
##   awayTeam = col_character(),
##   homeTeam = col_character(),
##   stadium = col_character(),
##   startTime = col_time(format = ""),
```

Reading in Data/Writing Out Data

```
scoreData
```

```
## # A tibble: 3,471 x 30
##    week  date   day season            awayTeam   AQ1   AQ2
##   <chr> <chr> <chr>  <int>               <chr> <int> <int>
## 1     1 5-Sep   Thu   2002 San Francisco 49ers     3     0
## 2     1 8-Sep   Sun   2002   Minnesota Vikings     3    17
## 3     1 8-Sep   Sun   2002  New Orleans Saints     6     7
## 4     1 8-Sep   Sun   2002        New York Jets     0    17
## 5     1 8-Sep   Sun   2002   Arizona Cardinals    10     3
## # ... with 3,466 more rows, and 21 more variables: AOT <int
## #   AFinal <int>, homeTeam <chr>, HQ1 <int>, HQ2 <int>, HQ3
## #   HQ4 <int>, HOT <int>, HOT2 <int>, HFinal <int>, stadium
## #   startTime <time>, toss <chr>, roof <chr>, surface <chr>
## #   duration <int>, attendance <chr>, weather <chr>, vegasL
## #   OU <chr>
```

Reading in Data/Writing Out Data

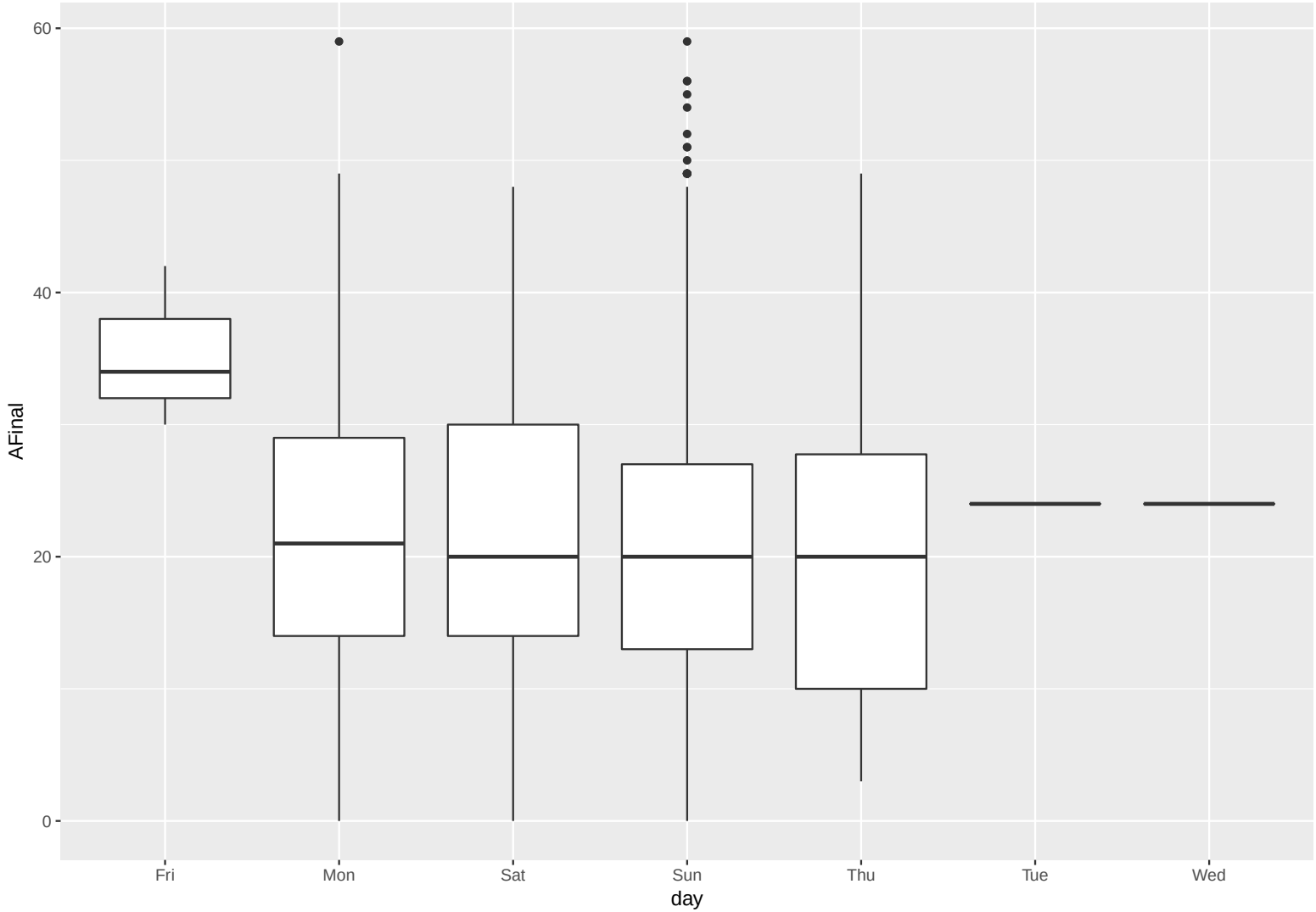
Notice: fancy printing!

- `tidyverse` data frames are special class (tbl_df or tibble)
- Printing method optimal

```
attributes(scoreData)$class
```

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

```
ggplot(data = scoreData, aes(x = day, y = AFinal)) +
    geom_boxplot() #easy plotting (covered later)!
```



Reading in Data/Writing Out Data

Reading in any delimited file

- Read in umps.txt file (a '>' delimited file)
- No column names provided in file
    - Year Month Day Home Away HPUmpire
- Use `read_delim()` (check help!)

```
umpData <- read_delim("datasets/umps2012.txt",
                      delim = ">",
     col_names = c("Year", "Month", "Day", "Home", "Away",
                                    "HPUmpire")
)
```

```
## Parsed with column specification:
## cols(
##   Year = col_integer(),
##   Month = col_integer(),
##   Day = col_integer(),
##   Home = col_character(),
##   Away = col_character(),
##   HPUmpire = col_character()
## )
```

Reading in Data/Writing Out Data

```
umpData
```

```
## # A tibble: 2,359 x 6
##    Year Month   Day  Home  Away          HPUmpire
##   <int> <int> <int> <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 in Data/Writing Out Data

Reading in any delimited file

- Functions from *readr* and their purpose

Delimiter	Function
comma ‘,’	read_csv()
tab	read_tsv()
space ‘’	read_table()
semi-colon ‘;’	read_csv2()
other	read_delim(. . . ,delim = ,. . .)

Reading in Data/Writing Out Data

Excel Data

- Read in censusEd.xls

Reading in Data/Writing Out Data

Excel Data

- Read in censusEd.xls
- Using `read_excel()` from `readxl` package
  - Reads both xls and xlsx files
  - Detects format from extension given
  - Specify sheet with name or integers (or NULL for 1st)

Reading in Data/Writing Out Data

Excel Data

- Read in censusEd.xls
- Using `read_excel()` from `readxl` package
  - Reads both xls and xlsx files
  - Detects format from extension given
  - Specify sheet with name or integers (or NULL for 1st)

```
library(readxl)
#just first sheet
edData <- read_excel("datasets/EDU01.xls", sheet = "EDU01A")
```

Reading in Data/Writing Out Data

```
edData
```

```
## # A tibble: 3,198 x 42
##       Area_name STCOU EDU010187F EDU010187D EDU010187N1 EDU
##           <chr> <chr>      <dbl>      <dbl>       <chr>
## 1 UNITED STATES 00000          0   40024299        0000
## 2       ALABAMA 01000          0     733735        0000
## 3   Autauga, AL 01001          0       6829        0000
## 4   Baldwin, AL 01003          0      16417        0000
## 5   Barbour, AL 01005          0       5071        0000
## # ... with 3,193 more rows, and 36 more variables: EDU01018
## #   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>
```

Reading in Data/Writing Out Data

Excel Data

- Using `read_excel()` from `readxl` package
  - Specify sheet with name or integers (or `NULL` for 1st)
  - Look at sheets available

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

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

Reading in Data/Writing Out Data

Excel Data

- Using `read_excel()` from readxl package
  - Specify cells with contiguous range

```
library(readxl)
#just first sheet
edData <- read_excel("datasets/censusEd.xls",
                                        sheet = "EDU01A",
                     range = cell_cols("A:D")
                     )
```

Reading in Data/Writing Out Data

```
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 `sheet` =name or integers
- Look at sheets available with `excel_sheets()`
- Specify cells with continguous range (`range = cell_cols("...")`)
- Specify cell with `range = "R1C2:R2C5"`

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

```
write_csv(x = edData, path = "datasets/output/edData.csv")
```

Recap/Other Packages/Functions

- Reading Data

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

- Write data with `write_csv()` from readr

Activity

- **Reading/Writing Data Activity** instructions available on web
- Feel free to work in small groups
- Feel free to ask questions about anything you didn't understand or the activity!