

## ENTERING / CLEANING DATA 1

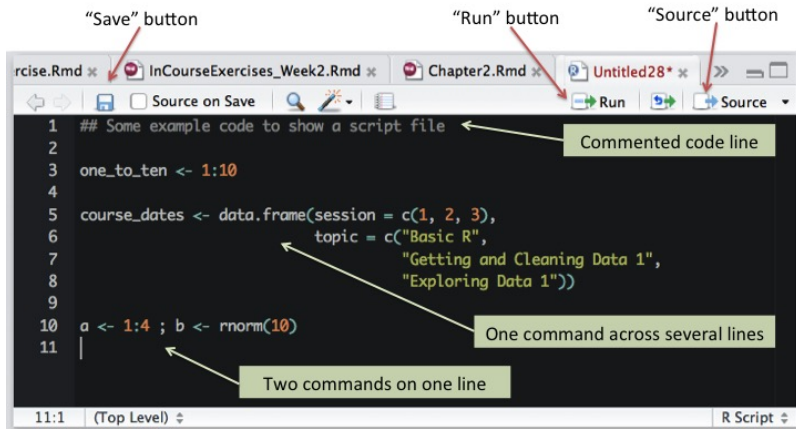
## R SCRIPTS

# R SCRIPTS

If you are writing code you think you will use later, write it in an *R script* file rather than using the console.

- Open a new script file in RStudio: File -> New File -> R Script.
- To run code from an R script file in RStudio, you can use the Run button (or Command-R). It will run whatever's on your cursor line or whatever's highlighted.
- To run the whole script, use `source`.
- Save scripts using the extension `.R`

# R SCRIPTS



## GETTING DATA INTO R

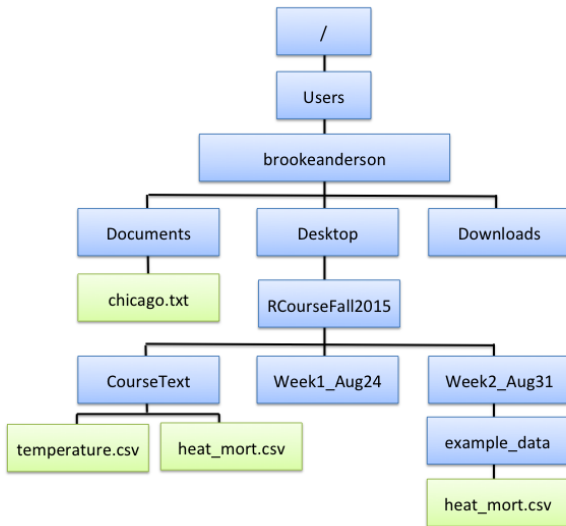
# BASICS OF GETTING DATA INTO R

Basic approach:

- Download data to your computer
- Make sure R is working in the directory with your data (`getwd`, `setwd`)
- Read data into R (`read.csv`, `read.table`)
- Check to make sure the data came in correctly (`dim`, `head`, `tail`, `str`)

## DIRECTORIES

# COMPUTER DIRECTORY STRUCTURE





# DIRECTORIES

You can check your working directory anytime using `getwd()`:

```
getwd()
```

```
## [1] "/Users/brookeanderson/RProgrammingForResearch/slides"
```

# DIRECTORIES

You can use `setwd()` to change your directory.

To get to your home directory (for example, mine is “/Users/brookeanderson”), you can use the abbreviation `~`.

For example, if you want to change into your home directory and print its name, you could run:

```
setwd("~/")  
getwd()
```

```
## [1] "/Users/brookeanderson"
```

# DIRECTORIES

The most straightforward way to read in data is often to put it in your working directory and then read it in using the file name. If you're working in the directory with the file you want, you should see the file if you list files in the working directory:

```
list.files()
```

```
## [1] "CourseNotes_Week1.pdf"
## [2] "CourseNotes_Week1.Rmd"
## [3] "CourseNotes_Week2.pdf"
## [4] "CourseNotes_Week2.Rmd"
## [5] "CourseOverview.pdf"
## [6] "CourseOverview.Rmd"
```

# GETTING AROUND DIRECTORIES

There are a few abbreviations you can use to represent certain relative or absolute locations when you're using `setwd()`:

Shorthand	Meaning
~	Home directory
.	Current working directory
..	One directory up from current working directory
../..	Two directories up from current working directory

# TAKING ADVANTAGE OF PASTE0

You can create an object with your directory name using `paste0`, and then use that to set your directory. We'll take a lot of advantage of this for reading in files.

The convention for `paste0` is:

```
## Generic code
[object name] <- paste0("[first thing you want to paste]",
                        "[what you want to add to that]",
                        "[more you want to add]")
```

# TAKING ADVANTAGE OF PASTE0

Here's an example:

```
my_dir <- paste0("~/RProgrammingForResearch",  
                "data/measles_data")
```

```
my_dir
```

```
## [1] "~/RProgrammingForResearchdata/measles_data"
```

```
setwd(my_dir)
```

# RELATIVE VERSUS ABSOLUTE PATHNAMES

When you want to reference a directory or file, you can use one of two types of pathnames:

- *Relative pathname*: How to get to the file or directory from your current working directory
- *Absolute pathname*: How to get to the file or directory from anywhere on the computer

# RELATIVE VERSUS ABSOLUTE PATHNAMES

Say your current working directory was  
/Users/brookeanderson/RProgrammingForResearch and you wanted  
to get into the subdirectory data. Here are examples using the two types  
of pathnames:

Absolute:

```
setwd("/Users/brookeanderson/RProgrammingForResearch/data")
```

Relative:

```
setwd("data")
```



## RELATIVE VERSUS ABSOLUTE PATHNAMES

Here are some other examples of relative pathnames:

If data is a subdirectory of your current parent directory:

```
setwd("../data")
```

If data is a subdirectory of your home directory:

```
setwd("~/data")
```

If data is a subdirectory of the subdirectory Ex of your current working directory:

```
setwd("Ex/data")
```

## READING DATA INTO R

# WHAT KIND OF DATA CAN YOU GET INTO R?

The sky is the limit. . .

- Flat files
- Files from other statistical packages (SAS, Excel, Stata, SPSS)
- Tables on webpages (e.g., the table near the end of this page)
- Data in a database (e.g., SQL)
- Really crazy data formats used in other disciplines (e.g., netCDF files from climate folks, MRI data stored in Analyze, NIfTI, and DICOM formats)
- Data through APIs (e.g., GoogleMaps, Twitter)
- Incredibly messy data using `scan` and `readLines`

# TYPES OF FLAT FILES

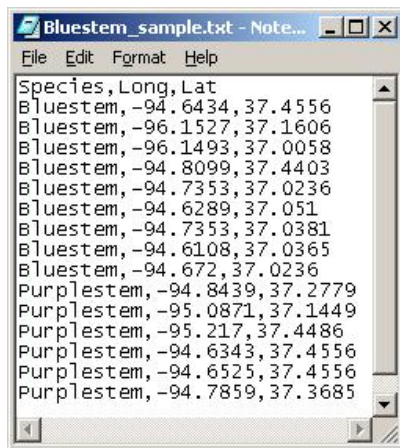
R can read in data from *a lot* of different formats. The only catch: you need to tell R how to do it.

To start, we'll look at flat files:

- ① Fixed width files
- ② Delimited files
  - “.csv”: Comma-separated values
  - “.tab”, “.tsv”: Tab-separated values
  - Other possible delimiters: colon, semicolon, pipe (“|”)

See if you can identify what types of files the following files are. . .

# WHAT TYPE OF FILE?



Bluestem\_sample.txt - Note...

File Edit Format Help

Species, Long, Lat

Bluestem, -94.6434, 37.4556

Bluestem, -96.1527, 37.1606

Bluestem, -96.1493, 37.0058

Bluestem, -94.8099, 37.4403

Bluestem, -94.7353, 37.0236

Bluestem, -94.6289, 37.051

Bluestem, -94.7353, 37.0381

Bluestem, -94.6108, 37.0365

Bluestem, -94.672, 37.0236

Purplestem, -94.8439, 37.2779

Purplestem, -95.0871, 37.1449

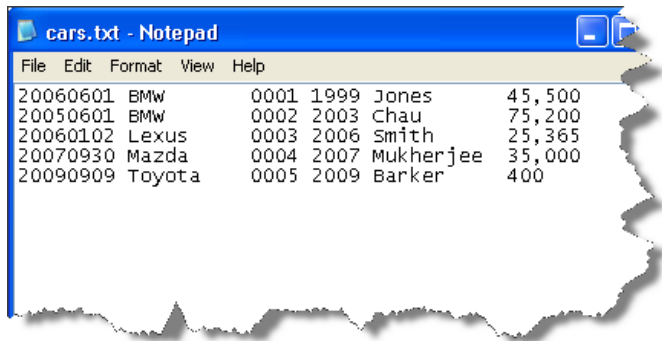
Purplestem, -95.217, 37.4486

Purplestem, -94.6343, 37.4556

Purplestem, -94.6525, 37.4556

Purplestem, -94.7859, 37.3685

# WHAT TYPE OF FILE?



# WHAT TYPE OF FILE?

```
H|20110606|pizza.txt|
D|10|Chicken Pesto|20|23|30|5.5|7.4|9.9|
D|10|Meatball|10|53|60|6.5|8.4|10.9|
D|10|Fire Cracker|3|13|60|5.8|7.9|11.9|
D|10|Spinach|1|2|5|5.5|7.0|8.8|
D|10|BBQ Chicken|35|102|95|6.5|7.9|10.9|
D|10|Vegetarian|5|13|28|4.5|7.9|9.5|
D|10|Mexican|11|33|36|5.5|7.4|9.9|
D|10|The Monaco|22|53|7|5.5|7.5|8.9|
D|10|Chilli Prawn|5|5|6|5.5|7.4|9.9|
D|10|Chefs Special|8|18|40|5.8|7.8|9.8|
D|10|Marinara|3|17|41|5.5|7.4|9.0|
D|10|Supreme|50|52|58|5.5|7.4|9.2|
D|10|Margherita|9|19|87|5.0|7.0|8.0|
D|10|Napoli|60|85|66|5.2|7.2|9.2|
D|10|Caprice|31|32|38|5.5|7.4|9.3|
D|10|Ham and Pineapple|18|39|28|5.8|7.0|9.0|
T|16|
```

# WHAT TYPE OF FILE?



```
File Edit Format View Help
Title Author Publisher ISBN
Harry Potter and the Sorcerer's Stone J.K. Rowling Arthur A. Levine Books 0590353403
The Da Vinci Code Dan Brown Doubleday 0385504209
Cracking The Da Vinci Code Simon Cox Barnes & Noble 0760759316
Illuminating Angels and Demons Simon Cox Barnes & Noble 0760767270
Bunnacula: A Rabbit-Tale of Mystery Deborah Howe Aladdin 0689806590
Bunnacula Strikes Again! James Howe Aladdin 0689814623
Red Dragon Thomas Harris Dutton Adult 0525945563
The Silence of the Lambs Thomas Harris St. Martin's Paperbacks 0312924585
I'm OK--You're OK Thomas Harris Harper Paperbacks 0060724277
```



# WHAT TYPE OF FILE?

File Edit Format View Help

```
Title,Subtitle,Larger work,Contributor #1,Contributor #2,Contributor #3,Contributor
#4,Genre,Publisher,Published Location,Date
Published,Instrumentation,Key,Location,Indiana Connection,Sheet Music
Consortium,Notes,Complete
""A"" You're Adorable",The alphabet song,,Buddy Kaye,Sidney Lippman,Fred Wise,,Popular
standard,Laurel Music Corporation,"New York, NY",1948,Voice and
piano/guitar or ukulele,C Major,,None,Yes,Perry Como pictured on cover,
"Aba Daba Honey Moon, The",,"""Two Weeks with Love"" Motion Picture",Arthur Fields,Walter
Donovan,,,"Popular Standard, Movie Selection",Leo Feist Inc.,,"New
York, NY",1942,Voice and Piano,C Minor,,None,Yes,,
Abi Bezunt,,,"Mamele"" Motion Picture",Abraham Ellstein,Molly Picon,,,"Popular standard,
Movie Selection",Metro Music Co.,,"New York, NY",1939,Voice and
Piano,E Minor,,None,No,Molly Picon pictured on cover,
Abdul the Bulbul Ameer,,Bob Kaai,Jim Smock,,,"Popular Standard,Calumet Music Co.
,"Chicago, IL",1935,"Voice, Piano, Hawaiian Guitar, Ukulele",G
Major,,None,Yes,Ben Pollack pictured on cover,
About A Quarter to Nine,,,"""Go Into Your Dance"" Motion Picture",Harry Warren,Al
Dubin,,,"Popular Standard, Movie Selection",M. Witmark & Sons,"New York,
NY",1935,"Voice, Piano, Guitar, Ukelele",E Minor,,None,No,Al Jolson and Ruby Keeler
pictured on cover,
Absent,,John. W. Metcalf,Catherine Young Glen,,,"Popular Standard,Arthur P.
Schmidt,"Boston, MA",1899,Voice and Piano,G Major,,None,Yes,,
The Academy Two-Step,,Barclay Walker,,,"Popular Standard,Carlin & Lennox,"Indianapolis,
IN",,"Piano,F Major,,Composer,No,,
Ac-cent-tchu-ate the Positive,Mister In Between,,,"""Here Come the Waves"" Motion
Picture",Harold Arlen,Johnny Mercer,,,"Popular Standard, Movie
Selection",Edwin H. Morris & Co.,,"New York, NY",1944,"Voice, Piano, Guitar",F
Major,,None,Yes,Bing Crosby and Betty Hutton pictured on cover,
Across the Alley From the Alamo,,Joe Greene,,,"Popular Standard,Leslie Music
```

# WHAT TYPE OF FILE?

1000233	Miralda	John
1000234	Faley	Nick
1000235	Baylog	Cathy
1000236	Gallardo	Mike
1000237	Christian	Daniel
1000238	Baufield	Daniel
1000239	Frazier	Robert
1000240	Garrido	Edward
1000241	Williams	Zachary
1000242	Morel	David
	Padilla	Damian
1000244	Rosenberg	Wayne
1000245	Blanchard	Phong S
1000246	Wiggins	David
1000247	Miller	Jeffrey
1000248	Coon	Terry
1000249	Chretien	Walter
1000250	Myers	Timothy
1000233	Miralda	John
1000234	Faley	Nick
1000235	Baylog	Cathy

# READING IN FLAT FILES

R can read any of these types of files using one of the `read.table` and `read.fwf` functions. Find out more about those functions with:

```
?read.table  
?read.fwf
```

## READ.TABLE FAMILY OF FUNCTIONS

Some of the interesting options with the `read.table` family of functions are:

Option	Description
<code>sep</code>	What is the delimiter in the data?
<code>skip</code>	How many lines of the start of the file should you skip?
<code>header</code>	Does the first line you read give column names?
<code>as.is</code>	Should you bring in strings as characters, not factors?
<code>nrows</code>	How many rows do you want to read in?
<code>na.strings</code>	How are missing values coded?

## READ.TABLE FAMILY OF FUNCTIONS

All members of the `read.table` family are doing the same basic thing. The only difference is what defaults they have for the separator (`sep`) and the decimal point (`dec`).

Members of the `read.table` family:

Function	Separator	Decimal point
<code>read.csv</code>	comma	period
<code>read.csv2</code>	semi-colon	comma
<code>read.delim</code>	tab	period
<code>read.delim2</code>	tab	comma

# READING IN ONLINE FLAT FILES

If you're reading in data from a non-secure webpage (i.e., one that starts with `http`), if the data is in a "flat-file" format, you can just read it in using the web address as the file name:

```
url <- paste0("http://www2.unil.ch/comparativegenometrics",  
              "/docs/NC_006368.txt")  
ld_genetics <- read.delim(url, header = TRUE)  
ld_genetics[1:5, 1:4]
```

```
##      pos  nA  nC  nG  
## 1    500 307 153 192  
## 2   1500 310 169 207  
## 3   2500 319 167 177  
## 4   3500 373 164 168  
## 5   4500 330 175 224
```

# READING IN ONLINE FLAT FILES

If you want to read in data from a secure webpage (e.g., one that starts with `https`), then you'll need to do something different.

First, you'll need to install then load the package `repmis`:

```
# install.packages("repmis")  
library(repmis)
```

## READING IN ONLINE FLAT FILES

Now you can use the `source.data` function to read in data from places like GitHub and Dropbox public folders:

```
url <- paste0("https://raw.githubusercontent.com/cmriivers/",  
              "ebola/master/country_timeseries.csv")  
ebola <- source_data(url)
```

```
## Downloading data from: https://raw.githubusercontent.com/cmri
```

```
## SHA-1 hash of the downloaded data file is:
```

```
## 6da83b3d2017245217d35989960184234a6c4e7f
```

```
ebola[1, 1:3]
```

```
##      Date Day Cases_Guinea
```

```
## 1 1/5/2015 289      2776
```



## SAVING DATA AS R OBJECTS

# SAVING R OBJECTS

You can save an R object you've created as an `.RData` file using `save()`:

```
save(ebola, file = "Ebola.RData")  
list.files()
```

```
## [1] "CourseNotes_Week1.pdf"  
## [2] "CourseNotes_Week1.Rmd"  
## [3] "CourseNotes_Week2.pdf"  
## [4] "CourseNotes_Week2.Rmd"  
## [5] "CourseOverview.pdf"  
## [6] "CourseOverview.Rmd"  
## [7] "Ebola.RData"
```

This saves to your current working directory (unless you specify a different location).

# LOADING R OBJECTS

Then you can re-load the object later using `load()`:

```
rm(ebola)
ls()
```

```
## [1] "dirpath_shortcuts" "ld_genetics"
## [3] "my_dir"            "url"
```

```
load("Ebola.RData")
ls()
```

```
## [1] "dirpath_shortcuts" "ebola"
## [3] "ld_genetics"       "my_dir"
## [5] "url"
```

# SAVING R OBJECTS

One caveat for saving R objects: some people suggest you avoid this if possible, to make your research more reproducible.

Imagine someone wants to look at your data and code in 30 years. R might not work the same, so you might not be able to read an `.RData` file. However, you can open flat files (e.g., `.csv`, `.txt`) and R scripts (`.R`) in text editors— you should still be able to do this regardless of what happens to R.

Potential exceptions:

- You have an object that you need to save that has a structure that won't work well in a flat file
- Your starting dataset is really, really large, and it would take a long time for you to read in your data fresh every time

## CLEANING UP DATA IN R

# RENAMING COLUMNS

Often, you'll want to change the column names of a dataframe as soon as you bring in the data, especially if the original ones have things like spaces. We'll look at this in the `icd10` data.

```
library(readxl)
```

```
## DEFINEDNAME: 20 00 00 01 0b 00 00 00 01 00 00 00 00 00 00 06  
## DEFINEDNAME: 20 00 00 01 0b 00 00 00 01 00 00 00 00 00 00 06  
## DEFINEDNAME: 20 00 00 01 0b 00 00 00 01 00 00 00 00 00 00 06  
## DEFINEDNAME: 20 00 00 01 0b 00 00 00 01 00 00 00 00 00 00 06
```

```
icd10 <- read_excel("icd-10.xls")
```

# RENAMING COLUMNS

Take a look at the `icd10` data we loaded in the exercise:

```
icd10[1:4, ]
```

```
##           Code
## 1 A00-B99
## 2 A00-A09
## 3      A00
## 4   A00.0
##
##                               ICD Title
## 1      I. Certain infectious and parasitic diseases
## 2                Intestinal infectious diseases
## 3                                Cholera
## 4 Cholera due to Vibrio cholerae 01, biovar cholerae
```

# RENAMING COLUMNS

You can use the `colnames()` function to find out what the current column names are:

```
colnames(icd10)
```

```
## [1] "Code"      "ICD Title"
```



# RENAMING COLUMNS

You can also **rename** column names using `colnames()`. You just put this call on the left of the assignment, and what you want to assign them on the left. The new names will need to be in a vector the same length as the number of columns.

```
colnames(icd10) <- c("code", "title")  
icd10[1:2, ]
```

```
##           code  
## 1 A00-B99  
## 2 A00-A09  
##  
##                                     title  
## 1 I. Certain infectious and parasitic diseases  
## 2               Intestinal infectious diseases
```

## USING SUBSET()

You will often want to use only a portion of your data. You can use `subset()` to create a subset, using logical operators. For example, if you wanted to just pull out the rows of the `icd10` dataframe that start with the letter “A”, you could run:

```
icd10_a <- subset(icd10, substr(code, 1, 1) == "A")
```

## USING SUBSET()

Here are the ends of the original and the subsetted dataframes:

```
tail(icd10, 2)
```

```
##           code
## 11089 Y89.9
## 11090  <NA>
##
##                                     title
## 11089 Sequelae of unspecified external cause
## 11090                                     <NA>
```

```
tail(icd10_a, 2)
```

```
##           code
## 458 A98.8
## 459  A99
##
##                                     title
## 458 Other specified viral hemorrhagic fevers
## 459  Unspecified viral hemorrhagic fever
```

# USING SUBSET()

The convention for `subset()` is:

```
subset([name of dataframe],  
       [logical statement describing which rows to keep],  
       select = [vector with the names of columns to keep])
```

# COMMON LOGICAL OPERATORS IN R

Operator	Meaning	Example
<code>==</code>	equals	<code>subset(df, city == "Los Angeles")</code>
<code>!=</code>	does not equal	<code>subset(df, city != "Los Angeles")</code>
<code>%in%</code>	is in	<code>subset(df, city %in% c("Los Angeles", "S"))</code>
<code>is.na()</code>	is NA	<code>subset(df, is.na(cases))</code>
<code>!is.na()</code>	is not NA	<code>subset(df, !is.na(cases))</code>
<code>&amp;</code>	and	<code>subset(df, city == "Los Angeles" &amp; !is.na(cases))</code>
<code> </code>	or	<code>subset(df, city == "Los Angeles"   !is.na(cases))</code>

# ADDING COLUMNS

In R, you can use the `$` operator after a dataframe to pull out one of it's columns. For example:

```
ca_measles <- read.delim("measles_data/02-09-2015.txt",  
                          header = FALSE,  
                          col.names = c("city", "count"))
```

```
head(ca_measles$count)
```

```
## [1] 6 20 2 4 2 34
```

## ADDING COLUMNS

You can take advantage of this to add new columns to an existing dataframe. For example, this data is from Feb. 9, 2015. We can add a column with the date to `ca_measles`:

```
ca_measles$date <- rep("02-09-2015", length = nrow(ca_measles))  
head(ca_measles, 3)
```

##	city	count	date
## 1	ALAMEDA	6	02-09-2015
## 2	LOS ANGELES	20	02-09-2015
## 3	City of Long Beach	2	02-09-2015

# ADDING COLUMNS

Two notes:

- The previous example uses the `rep()` function, which will repeat a value `length` number of times
- If the value you assign to the new column is not the right length (remember, all columns in a dataframe must be vectors of equal length), R will try to “recycle” it to fill up the dataframe. So, the following call would have been a simpler alternative:

```
ca_measles$date <- "02-09-2015"  
ca_measles[1:2, ]
```

```
##           city count      date  
## 1      ALAMEDA      6 02-09-2015  
## 2 LOS ANGELES     20 02-09-2015
```



## DATE CLASS

One final common task in cleaning data is to change the class of some of the columns. This is especially common for dates, which will usually be read in as characters or factors.

# VECTOR CLASSES

Here are a few common vector classes in R:

Class	Example
character	"Chemistry", "Physics", "Mathematics"
numeric	10, 20, 30, 40
factor	Male [underlying number: 1], Female [2]
Date	"2010-01-01" [underlying number: 14,610]
logical	TRUE, FALSE

# VECTOR CLASSES

To find out the class of a vector, you can use `class()`:

```
class(ca_measles$date)
```

```
## [1] "character"
```

# VECTOR CLASSES

To find out the classes of all columns in a dataframe, you can use `str()`:

```
str(ca_measles)
```

```
## 'data.frame':    13 obs. of  3 variables:
## $ city : Factor w/ 13 levels "ALAMEDA","City of Long Beach",
## $ count: int  6 20 2 4 2 34 5 6 13 3 ...
## $ date : chr  "02-09-2015" "02-09-2015" "02-09-2015" "02-09-
```

# CONVERTING TO DATE CLASS

To convert a vector to the Date class, you can use `as.Date()`:

```
ca_measles$date <- as.Date(ca_measles$date,  
                           format = "%m-%d-%Y")  
head(ca_measles$date, 3)
```

```
## [1] "2015-02-09" "2015-02-09" "2015-02-09"
```

```
class(ca_measles$date)
```

```
## [1] "Date"
```

## CONVERTING TO DATE CLASS

Once you have an object in the Date class, you can do things like plot by date, calculate the range of dates, and calculate the total number of days the dataset covers:

```
range(ca_measles$date)
```

```
## [1] "2015-02-09" "2015-02-09"
```

```
diff(range(ca_measles$date))
```

```
## Time difference of 0 days
```

# CONVERTING TO DATE CLASS

The only tricky thing is learning the abbreviations for the format option. Here are some common ones:

Abbreviation	Meaning
%m	Month as a number (e.g., 1, 05)
%B	Full month name (e.g., August)
%b	Abbreviated month name (e.g., Aug)
%y	Two-digit year (e.g., 99)
%Y	Four-digit year (e.g., 1999)

# CONVERTING TO DATE CLASS

Here are some examples:

Your date	format =
10/23/2008	"%m/%d%Y"
08-10-23	"%y-%m-%d"
Oct. 23 2008	"%b. %d %Y"
October 23, 2008	"%B %d, %Y"