Entering / cleaning data 1

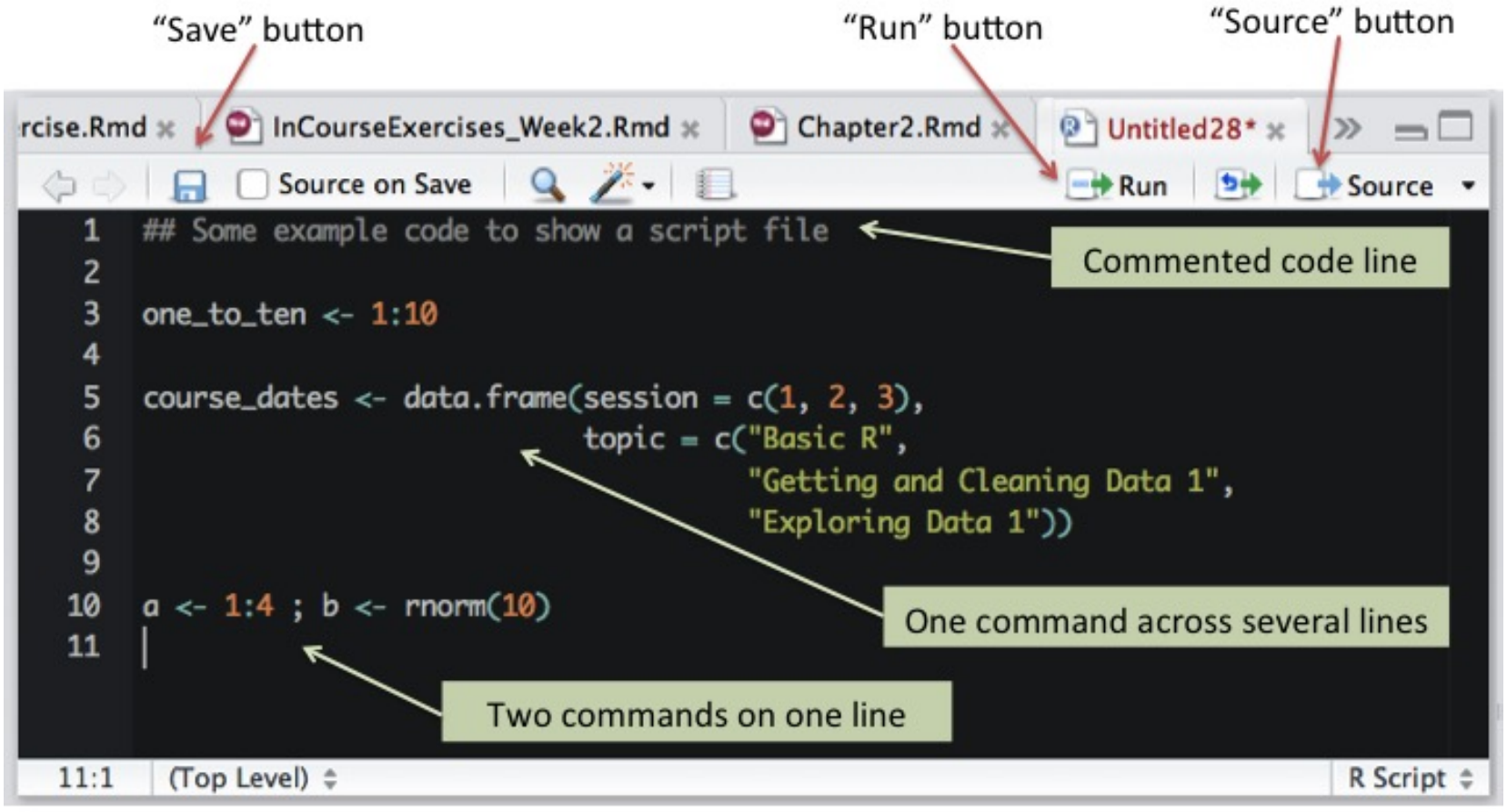
R scripts

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


"Save" button
"Run" button
"Source" button
rcise.Rmd
InCourseExercises_Week2.Rmd
Chapter2.Rmd
Untitled28*
Source on Save
Run
Source
1 ## Some example code to show a script file
2
3 one_to_ten <- 1:10
4
5 course_dates <- data.frame(session = c(1, 2, 3),
6 topic = c("Basic R",
7 "Getting and Cleaning Data 1",
8 "Exploring Data 1"))
9
10 a <- 1:4 ; b <- rnorm(10)
11
Commented code line
One command across several lines
Two commands on one line
11:1
(Top Level)
R Script

Getting data into R

Basics of getting data into R

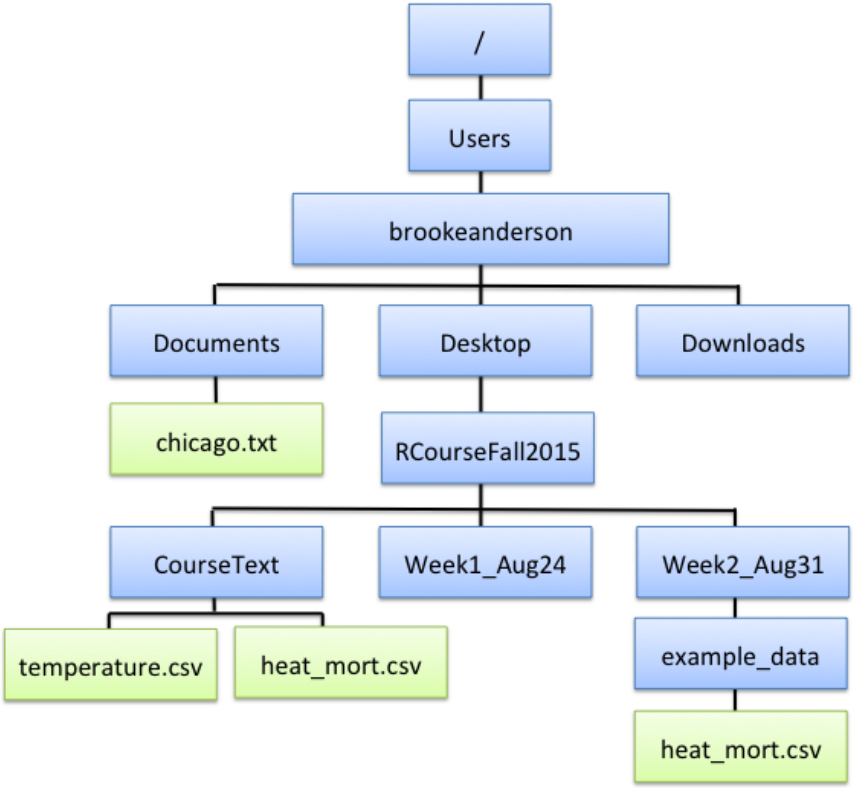
Basic approach:

- 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

Directories

Anytime you work in R, R will run from within a directory somewhere on your computer. Let's review directories:



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:

```
[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("~/Desktop/RCourseFall2015/",
                 "Week2_Aug31")
my_dir
```

```
## [1] "~/Desktop/RCourseFall2015/Week2_Aug31"
```

```
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)
- Incrediably messy data using `scan` and `readLines`

Types of flat files

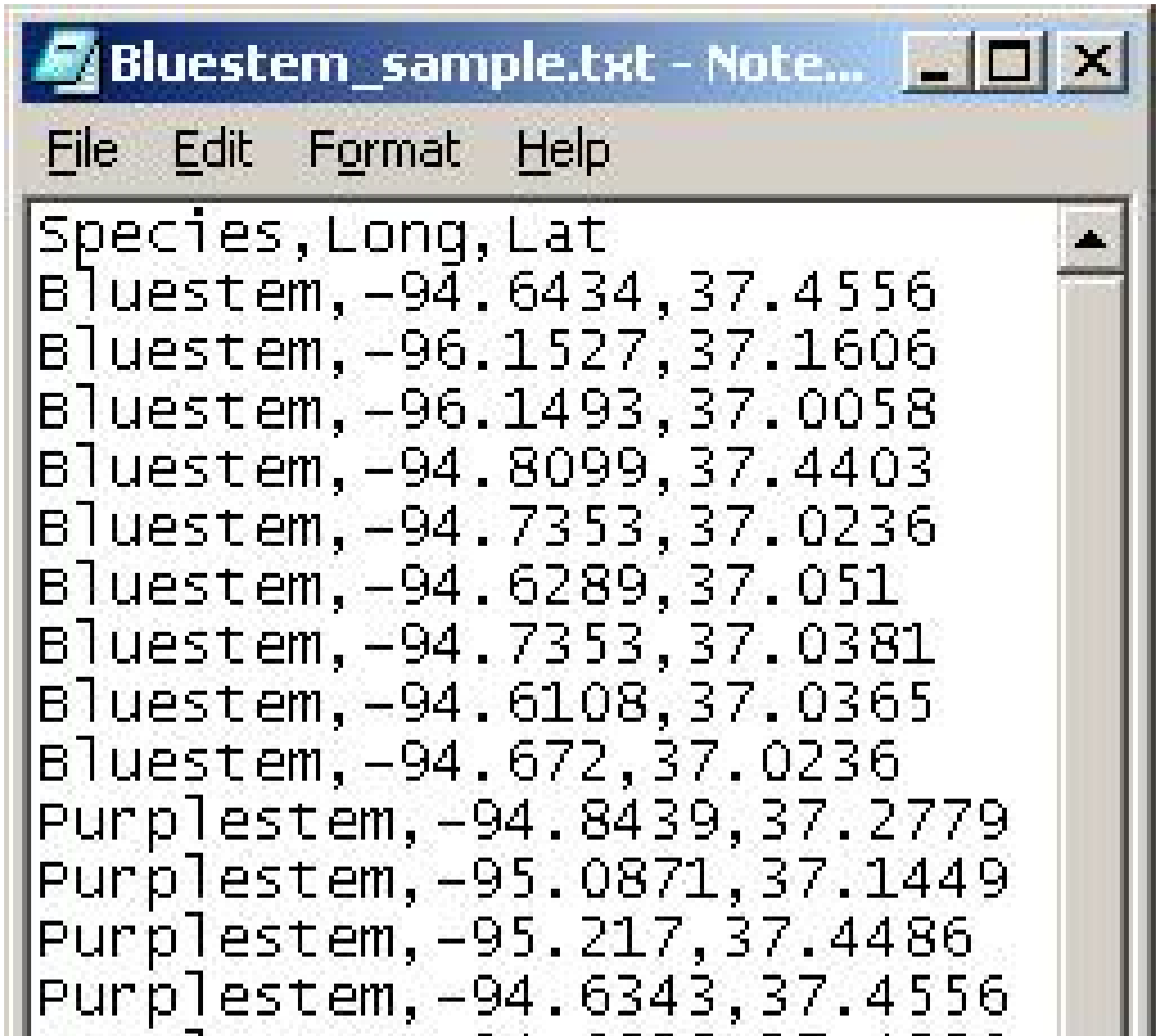
R can read in data from *a lot* of different formats. The only catch: you need to tell R how to do it.
To start, we'll look at flat files:

1. Fixed width files
2. Delimited files

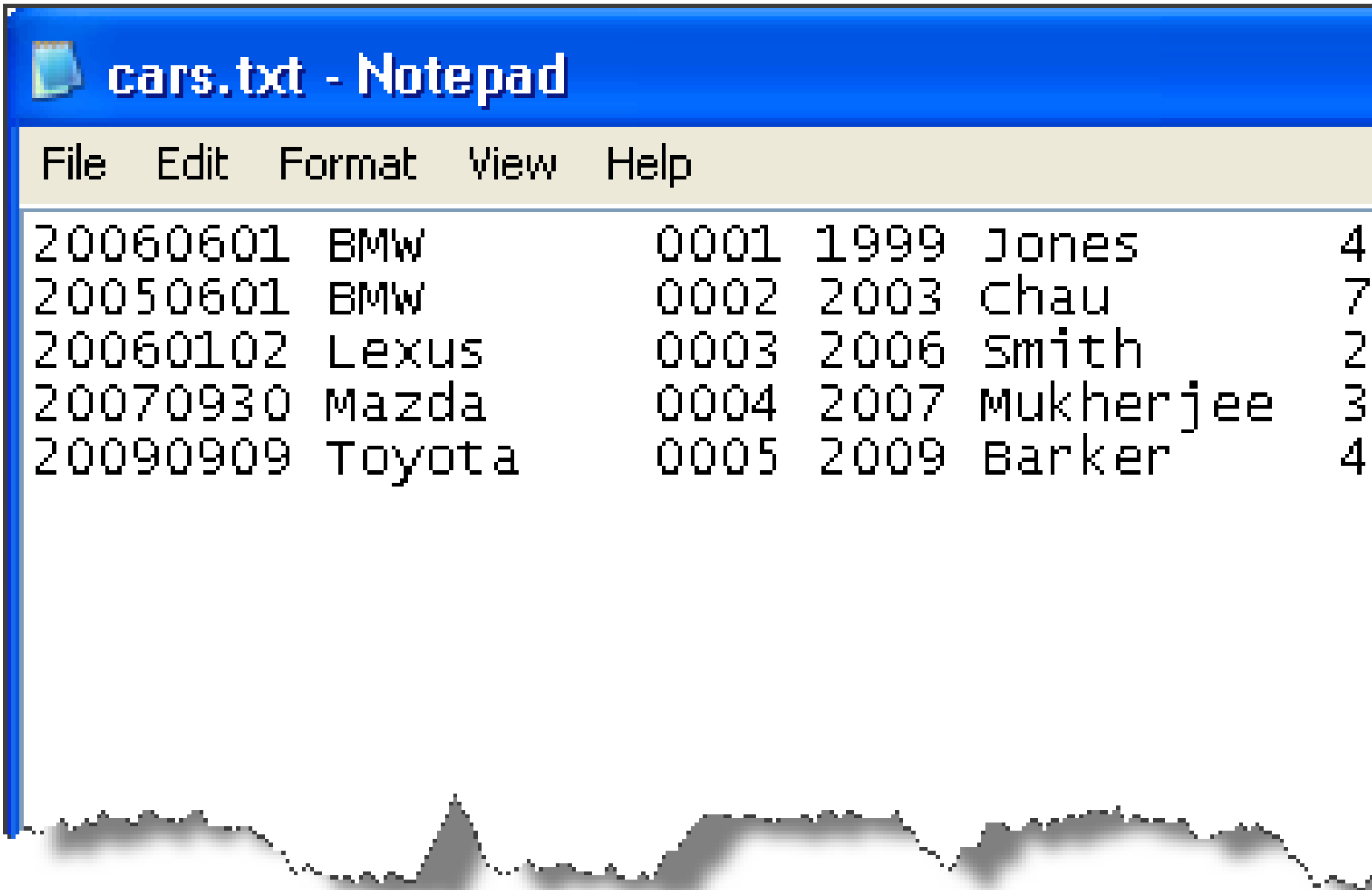
- ".csv": Comma-separated values
- ".tab", ".tsv": Tab-separated values
- Other possible delimiters: colon, semicolon, pipe ("|")

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

What type of file?



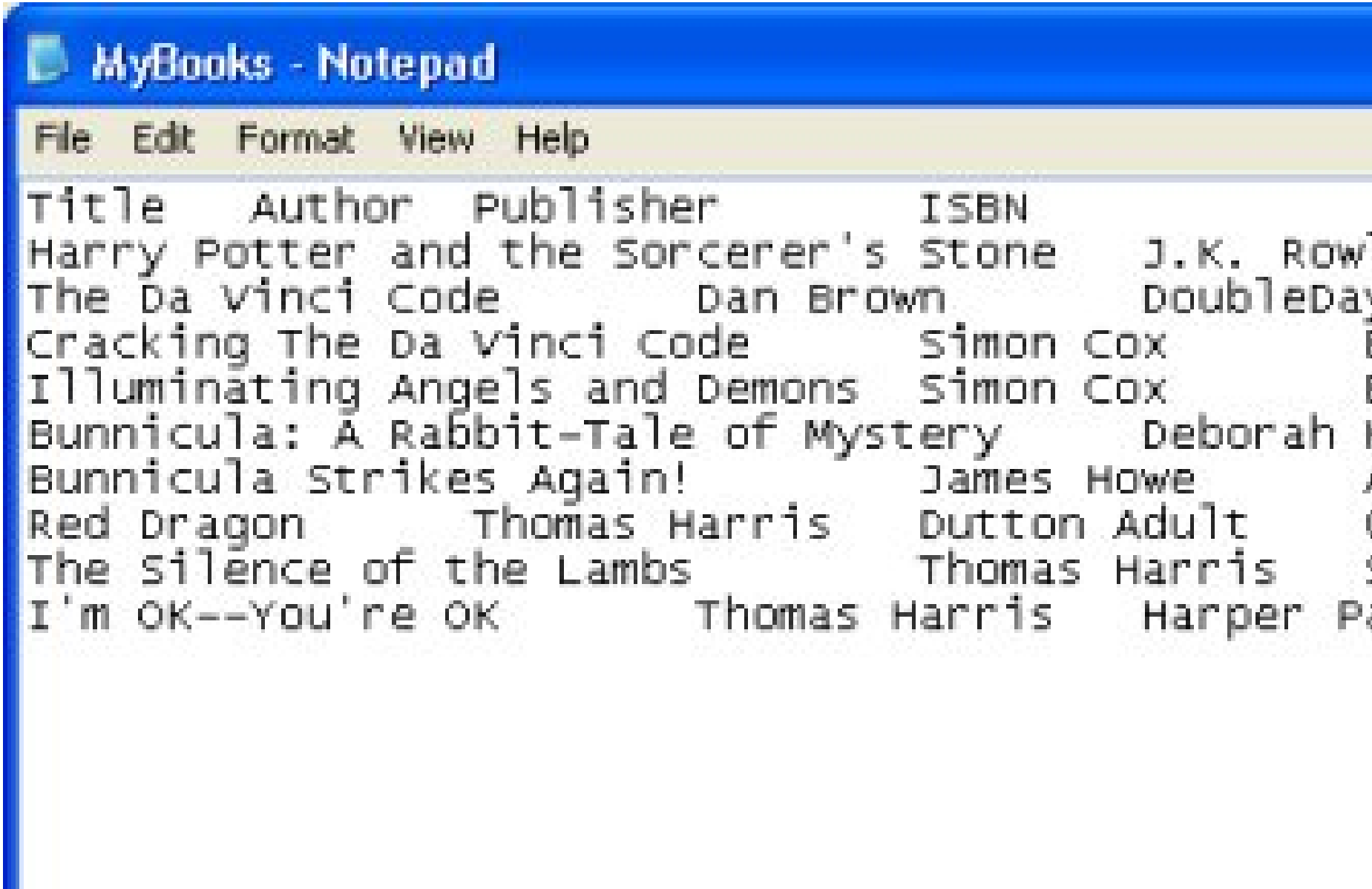
What type of file?



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?



What type of file?

File Edit Format View Help

```
Title,Subtitle,Larger Work,Contributor #1,Contr
#4,Genre,Publisher,Published Location,Date
Published,Instrumentation,Key,Location,Indiana
Consortium,Notes,Complete
""""A"" You're Adorable",The alphabet song,,Buddy
standard,Laurel Music Corporation,"New York, NY
piano/guitar or ukulele,C Major,,None,Yes,Perry
"Aba Daba Honey Moon, The",,""""Two Weeks with L
Donovan,,,"Popular Standard, Movie Selection",L
York, NY",1942,Voice and Piano,C Minor,,None,Ye
Abi Bezunt,,""""Mamele"" Motion Picture",Abraham
Movie Selection",Metro Music Co.,"New York, NY"
Piano,E Minor,,None,No,Molly Picon pictured on
Abdul the Bulbul Ameer,,,Bob Kaai,Jim Smock,,P
,"Chicago, IL",1935,"Voice, Piano, Hawaiian Gui
Major,,None,Yes,Ben Pollack pictured on cover,
About A Quarter to Nine,,""""Go Into Your Dance"
```

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   Conn         Terry
```

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
`sep`	What is the delimiter in the data?
`skip`	How many lines of the start of the file should you skip?
`header`	Does the first line you read give column names?
`as.is`	Should you bring in strings as characters, not factors?
`nrows`	How many rows do you want to read in?
`na.strings`	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
`read.csv`	comma	period
`read.csv2`	semi-colon	comma
`read.delim`	tab	period
`read.delim2`	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/cmrivers/",
              "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:
## 6da83b3d2017245217d3598996018423 4a6c4e7f
```

```
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
`==`	equals	`subset(df, city == "Los Angeles")`
`!=`	does not equal	`subset(df, city != "Los Angeles")`
`%in%`	is in	`subset(df, city %in% c("Los Angeles", "S`
`is.na()`	is NA	`subset(df, is.na(cases))`
`!is.na()`	is not NA	`subset(df, !is.na(cases))`
`&`	and	`subset(df, city == "Los Angeles" & !is.n`
`|`	or	`subset(df, city == "Los Angeles" | !is.n`

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"