CSV files

Andrew Ba Tran

Contents

This is from the second chapter of learn.r-journalism.com.

Comma separated files are the most common way to save spreadsheets that doesn't require a paid program from Microsoft to open.

What a csv file looks like

CSV file names end with a **.csv**

What a csv file looks like on the inside

This explains the values separated with commas part of the file name.

Importing CSV files

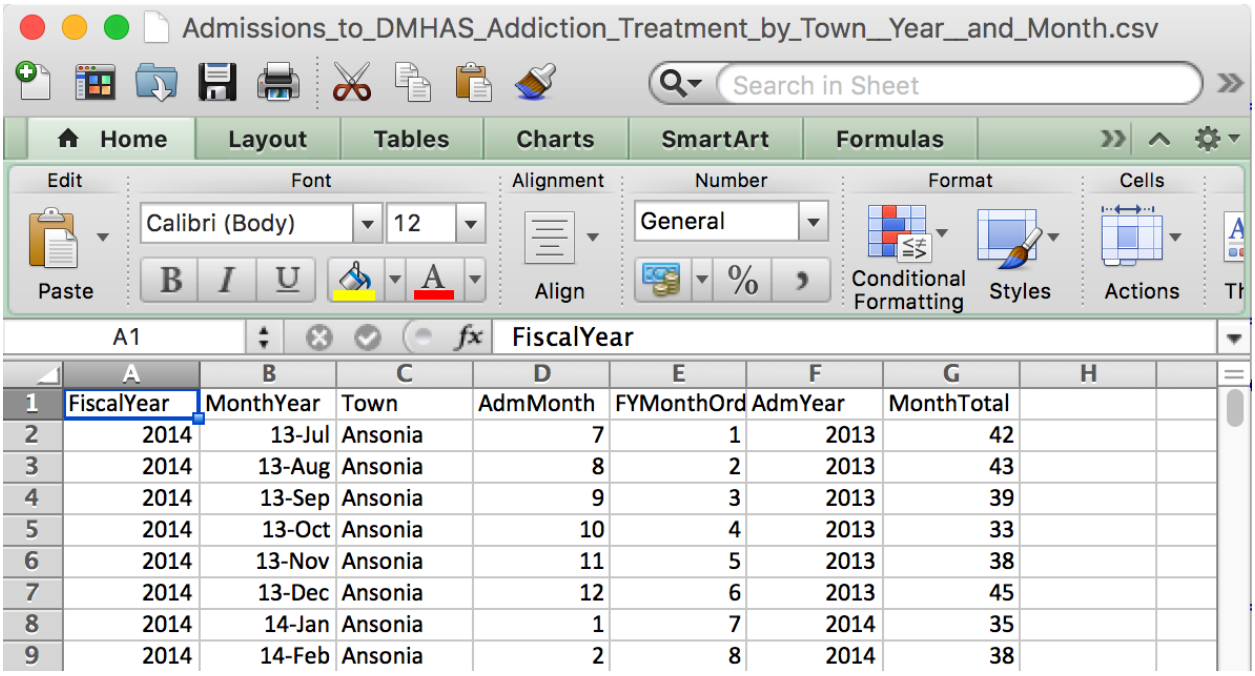
- Importing CSV is part of base R, no package needed
- But we're going to use a package anyway, **readr**

Two ways to get data

- If you have the URL address
- If you have the file on your computer

Get the URL

If you have the link to a CSV file, right click the link of the data and click **Copy Link Address**. This data set can be found on the Connecticut Open Data Portal.



	A	B	C	D	E	F	G	H
1	FiscalYear	MonthYear	Town	AdmMonth	FYMonthOrd	AdmYear	MonthTotal	
2	2014	13-Jul	Ansonia	7	1	2013	42	
3	2014	13-Aug	Ansonia	8	2	2013	43	
4	2014	13-Sep	Ansonia	9	3	2013	39	
5	2014	13-Oct	Ansonia	10	4	2013	33	
6	2014	13-Nov	Ansonia	11	5	2013	38	
7	2014	13-Dec	Ansonia	12	6	2013	45	
8	2014	14-Jan	Ansonia	1	7	2014	35	
9	2014	14-Feb	Ansonia	2	8	2014	38	

Figure 1:

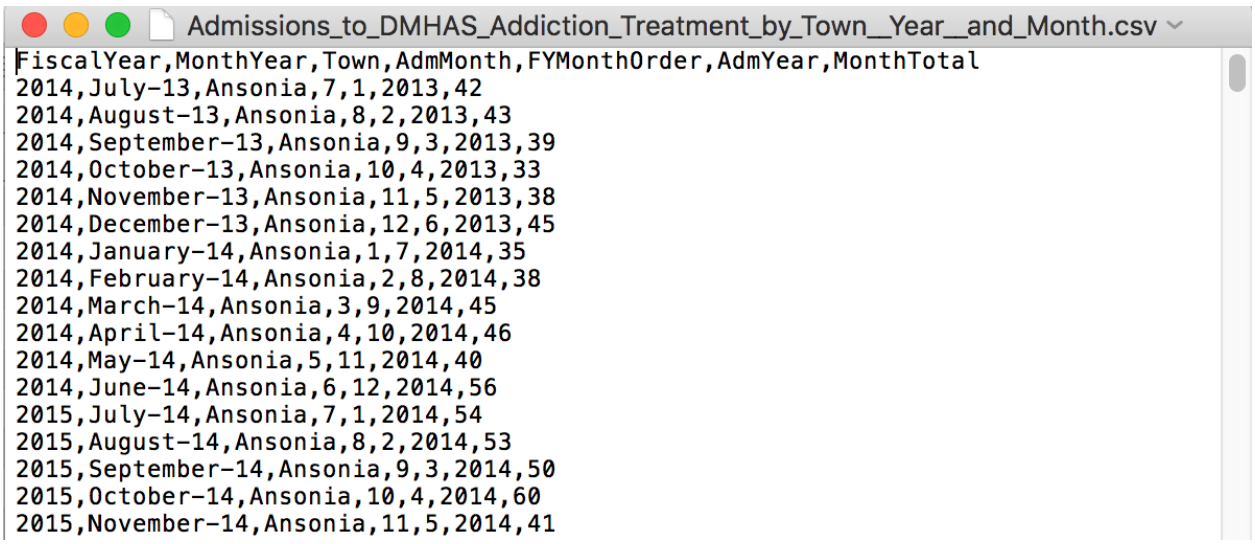


Figure 2:

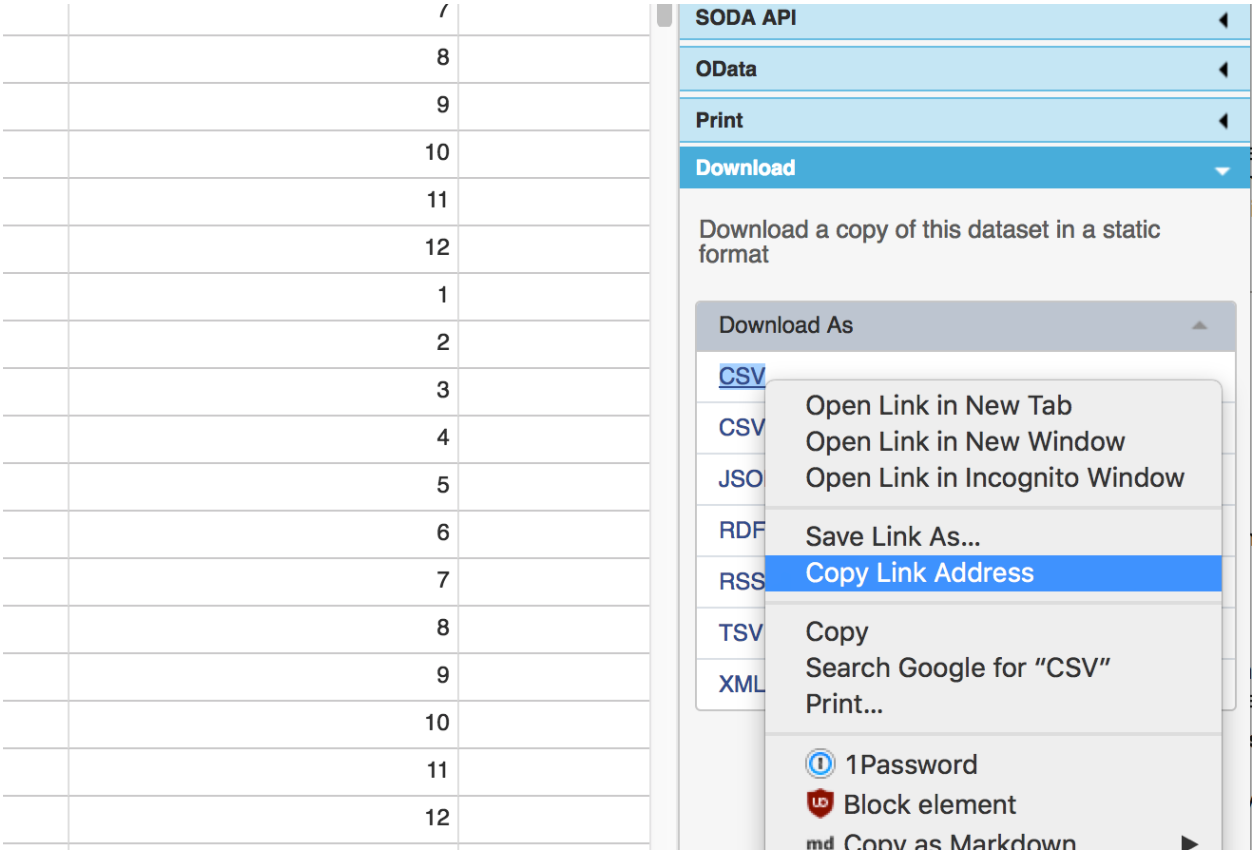

7
8
9
10
11
12
1
2
3
4
5
6
7
8
9
10
11
12
SODA API
OData
Print
Download
Download a copy of this dataset in a static format
Download As
CSV
CSV
JSO
RDF
RSS
TSV
XML
Open Link in New Tab
Open Link in New Window
Open Link in Incognito Window
Save Link As...
Copy Link Address
Copy
Search Google for "CSV"
Print...
1Password
Block element
Copy as Markdown

Figure 3:

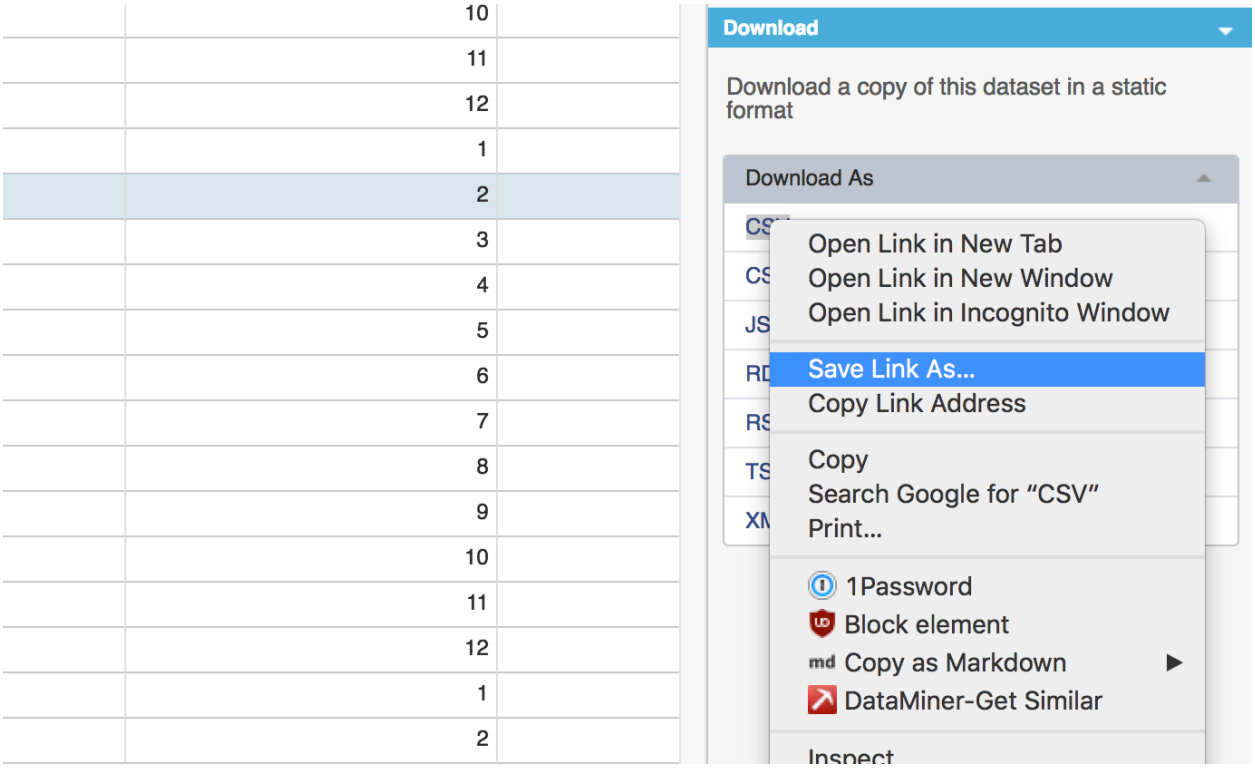


Figure 4:

read.csv()

The Base R function to import a CSV file is `read.csv()`. Just put the URL address in quotation marks and add the `stringsAsFactors=F` (In this code we're using the function `head()` which looks at the first 10 rows of data by default. We could pass it a number to change the number of rows to display)

```
df_csv <- read.csv("https://data.ct.gov/api/views/iyru-82zq/rows.csv?accessType=DOWNLOAD",
                   stringsAsFactors=F)
head(df_csv)
```

```
##   FiscalYear    MonthYear    Town AdmMonth FYMonthOrder AdmYear MonthTotal
## 1       2014      July-13 Ansonia        7            1    2013         42
## 2       2014    August-13 Ansonia        8            2    2013         43
## 3       2014 September-13 Ansonia        9            3    2013         39
## 4       2014   October-13 Ansonia       10            4    2013         33
## 5       2014  November-13 Ansonia       11            5    2013         38
## 6       2014  December-13 Ansonia       12            6    2013         45
```

The other way to import the data: Download it

When you right click on the link, instead of clicking **Copy Link Address**– this time, click **Save Link As...**

Save to your working directory.

After saving to the directory, click on the circle arrow on the right to refresh the files to make sure it's there.

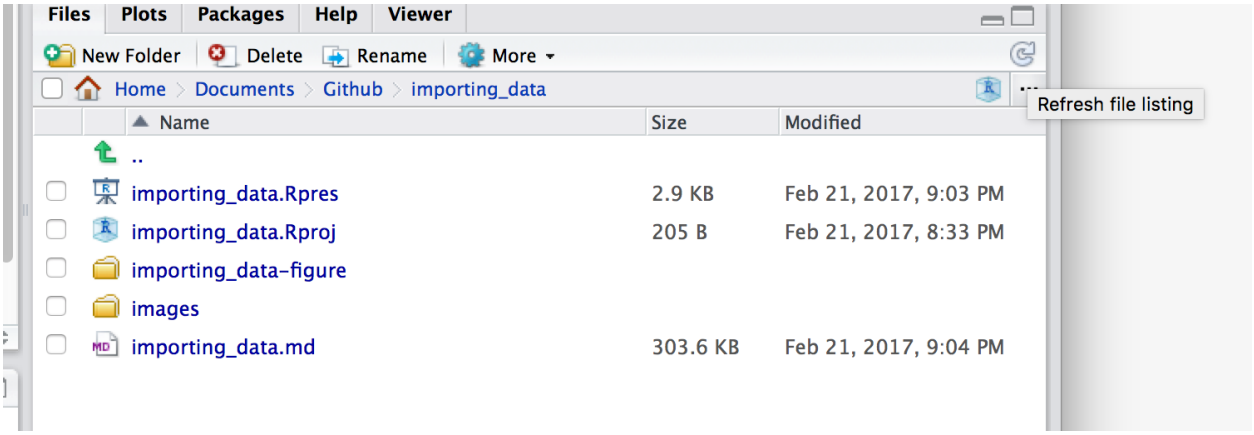


Figure 5:

Recall: How to change directories in RStudio

Either by typing `setwd("/directory/where/you/want")` or by clicking in the menu up top **Session > Set Working Directory > Choose Directory...**

Importing local csv data

Just like before, except instead of the URL, it's the name of the file.

Note: This will only work if the working directory is set to where the csv file is.

```
df_csv <- read.csv("data/Admissions_to_DMHAS_Addiction_Treatment_by_Town__Year__and_Month.csv",
                   stringsAsFactors=F)
```

stringsAsFactors=F

Why?

Blame statisticians.

Back when R was created the users weren't using it as we use it now, with all these different strings.

What happens when you don't use `stringsAsFactors=F`

```
df_csv <- read.csv("data/Admissions_to_DMHAS_Addiction_Treatment_by_Town__Year__and_Month.csv")
str(df_csv)
```

```
## 'data.frame':    3420 obs. of  7 variables:
##  $ FiscalYear  : int  2014 2014 2014 2014 2014 2014 2014 2014 2014 2014 ...
##  $ MonthYear   : Factor w/ 36 levels "April-14","April-15",..: 16 4 34 31 28 7 13 10 22 1 ...
##  $ Town        : Factor w/ 102 levels "Ansonia","Berlin",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ AdmMonth    : int  7 8 9 10 11 12 1 2 3 4 ...
##  $ FYMonthOrder: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ AdmYear     : int  2013 2013 2013 2013 2013 2013 2014 2014 2014 2014 ...
##  $ MonthTotal  : int  42 43 39 33 38 45 35 38 45 46 ...
```

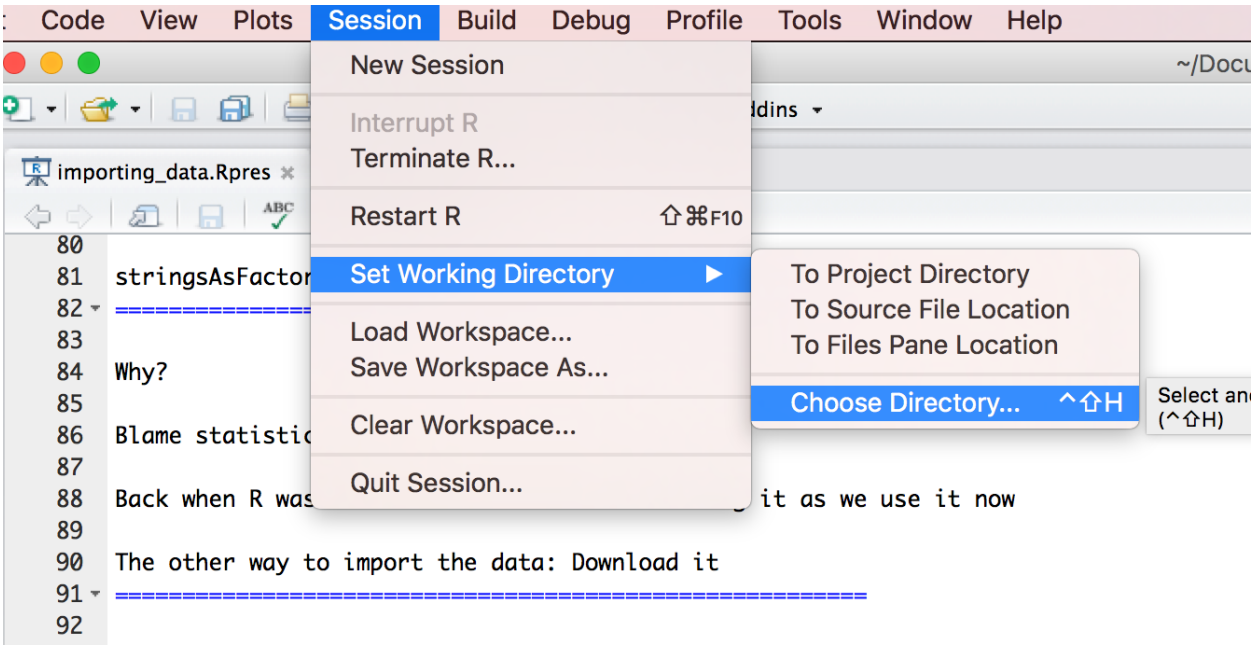


Figure 6:

Using read_csv() from the readr package

readr is a package that read rectangular data quickly and assumes characters are strings and not factors by default.

```
## If you don't have readr installed yet, uncomment and run the line below
#install.packages("readr")
library(readr)
df_csv <- read_csv("data/Admissions_to_DMHAS_Addiction_Treatment_by_Town__Year__and_Month.csv")
```

```
## Parsed with column specification:
## cols(
##   FiscalYear = col_integer(),
##   MonthYear = col_character(),
##   Town = col_character(),
##   AdmMonth = col_integer(),
##   FYMonthOrder = col_integer(),
##   AdmYear = col_integer(),
##   MonthTotal = col_integer()
## )
```

As you can see, the `read_csv()` function interpreted the **MonthYear** and **Town** columns as characters and not as Factors as `read.csv()` did.

Exporting CSV files

When you're done analyzing or transforming your data, you can save your dataframe as a CSV file with `write_csv()` from the **readr** package.

```r
# Pass the write_csv() function the name of the dataframe and what you want to call the file
write_csv(df_csv, "transformed_data.csv")
```

The file will save to your working directory, but you can specify sub directories with the function.

```r
# Pass the write_csv() function the name of the dataframe and what you want to call the file
write_csv(df_csv, "data/transformed_data.csv")
```

Exporting data frames with `NA`

Weird quirk alert: Exported files will include `NA`s so to strip them out, pass the variable `na="whatever"`.

```r
# This replaces the NAs with blanks
write_csv(df_csv, "data/transformed_data.csv", na="")
```