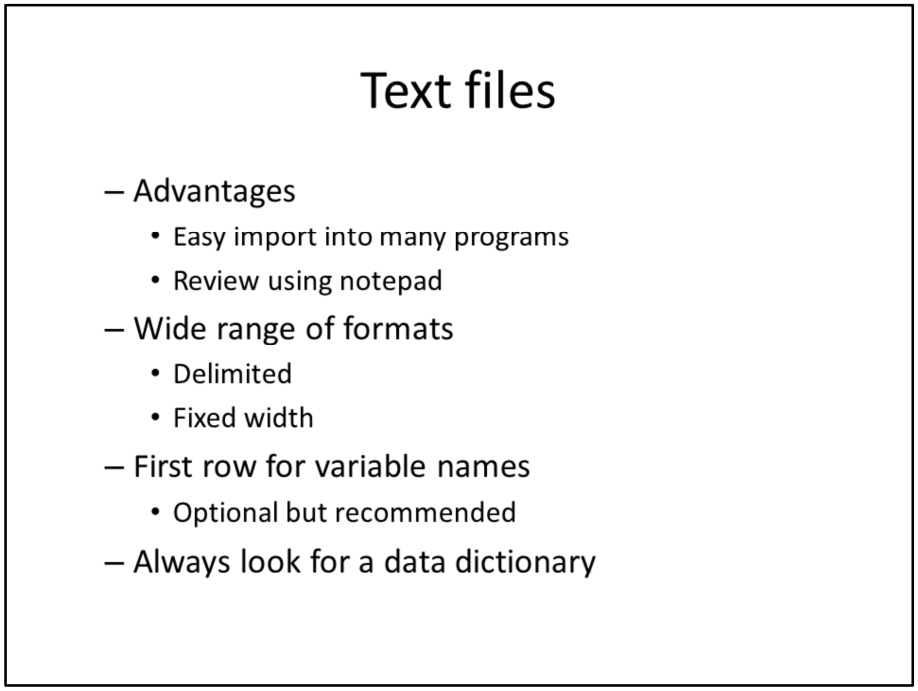
Module03: Reading text files

Steve Simon

Created 2020-02-08



One of the most important skils you will learn in this class is how to read data from text files. Text files are commonly used for data storage because they are easily imported into a variety of different programs. You can often peek at a text file using a simple program like Notepad in order to get a quick feel for what the data looks like.

Text files come in a variety of format. A delimited file uses a special character, often a comma, to designiate where one data value ends and the next one begins. In contrast, a fixed width file requires every variable to occupy a particular column or columns.

Many text files include the names of the variables in the first row of text. This is not required, but it is strongly recommended.

Read in comma separated values

- Type the following into notepad.

```
"x","y"
1,4
2,8
3,12
4,16
```

- Save it
  - in the dat directory
  - filename: simple.csv

In notepad or another text editor, type in the following values.

x,y

1,4

2,8

3,12

4,16

and save the file using the name simple.csv. Note the directory that you are saving it in. Ideally, it should be in a folder named “dat.”

the read.csv function

```
fn <- "../dat/simple.csv"
raw_data <- read.csv(file=fn,
  stringsAsFactors=FALSE)
raw_data
##   x  y
## 1 1  4
## 2 2  8
## 3 3 12
## 4 4 16
```

Comma separated files are so common that R has several dedicated functions for reading them. The simplest function is read.csv. It The best fuctions are in the readr library.

We will not talk about it in this module, but you almost always want to use the option stringsAsFactors=FALSE.

Space delimited files

– Type the following into notepad.

```
x y
1 4
2 8
3 12
4 16
```

– Save it

- in the dat directory
- filename: simple.txt

Now try something different. Type in the same data, but use a single space between each number rather than a comma.

Using the read.table function

```
fn <- "../dat/simple.txt"
raw_data <- read.table(fn, header=TRUE,
stringsAsFactors=FALSE, sep=" ")
raw_data
##   x  y
## 1 1  4
## 2 2  8
## 3 3 12
## 4 4 16
```

Review the options for the read.table command.

Type in the same data, but using the tilde (~) as a separator (e.g., 1~4). Save the file as tilde.txt.

Here is how you would use read.table to read this file.

Tab delimited files

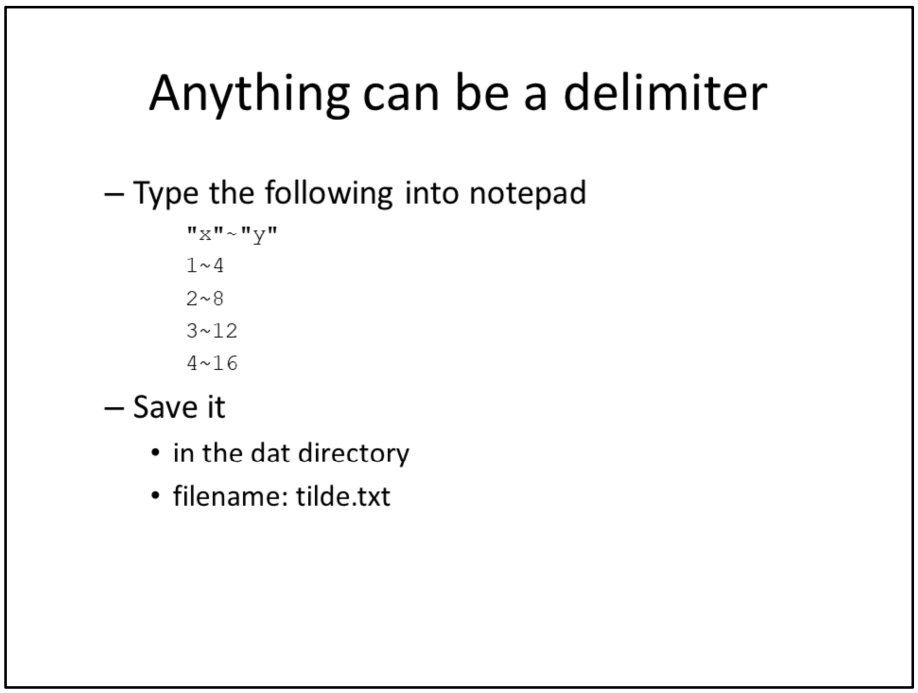
– Type the following into notepad, but everywhere you see a space, press the tab key instead of the space bar.

```
x   y
1   4
2   8
3   12
4   16
```

– Save it
  - in the dat directory
  - filename: simple.tsv

Using the read.table function

```
fn <- "../dat/simple.tsv"
raw_data <- read.table(fn, header=TRUE,
stringsAsFactors=FALSE, sep="\t")
raw_data
##   x  y
## 1 1  4
## 2 2  8
## 3 3 12
## 4 4 16
```



Type in the same data, but using the tilde (~) as a separator (e.g., 1~4). Save the file as tilde.txt.

Here is how you would use read.table to read this file.

Using the read.table function

```
fn <- "../dat/tilde.txt"
raw_data <- read.table(fn, header=TRUE,
stringsAsFactors=FALSE, sep="~")
raw_data
##   x  y
## 1 1  4
## 2 2  8
## 3 3 12
## 4 4 16
```

The data reads in just fine. Why would you use a tilde as a delimiter. Sometimes your data itself includes delimiters like spaces and commas, and then you might want to choose an obscure out of the way symbol to serve as a delimiter.

Reading fixed width format files

– Type the following into notepad
  - Space between the 1 and 4
  - Space between the 2 and 8
  - No space between the 3 and 12
  - No space between the 4 and 16

```
1 4
2 8
312
416
```

– Save it
  - in the dat directory
  - filename: fixed.txt

Here's another variation.

Sometimes you will get a file with no delimiters to save space. This requires that each variable takes up a fixed number of columns and that information is often specified in a separate file.

Create a fixed width file in notepad with the following lines.

1 4

2 8

312

416

and store it as fixed.txt.

The read.fwf function

```
fn <- "../dat/fixed.txt"
raw_data <- read.fwf(fn, c(1, 2))
raw_data
##   V1 V2
## 1  1  4
## 2  2  8
## 3  3 12
## 4  4 16
```

You should also consider the read.fortran function. Check out the help file.

Writing text files

- Similar structure to read functions
  - write.csv
  - write.table

If you want to write a text file, you can use either the write.csv or write.table functions. Look up the help files on these functions. We'll create a small data frame and store it in a comma delimited and tab delimited format.

Writing text files

```
raw_data <- data.frame(x=c(1, 2, 3, 4), y=c(4, 8,
12, 16))
fn <- "../results/output_data.txt"
write.table(raw_data, fn, row.names=FALSE)
```

Open these files in notepad to see what they look like.

Writing text files

```
## "x" "y"
## 1 4
## 2 8
## 3 12
## 4 16
```

If you opened up the data in notepad or some other program, you would see a space delimited file. You can change the delimiter, among other things

Example #1

– https://stats.idre.ucla.edu/stat/data/binary.csv

```
## admit,gre,gpa,rank
## 0,380,3.61,3
## 1,660,3.67,3
## 1,800,4,1
## 1,640,3.19,4
## 0,520,2.93,4
```

Here is the first real-world example. Peeking at the first six lines of data, you can see clearly that it is uses comma delimiter and the first row of data contains the variable names.

Example #1

- No formal data dictionary, but here is a description
  - "This dataset has a binary response (outcome, dependent) variable called admit. There are three predictor variables: gre, gpa and rank. We will treat the variables gre and gpa as continuous. The variable rank takes on the values 1 through 4. Institutions with a rank of 1 have the highest prestige, while those with a rank of 4 have the lowest."
  - Description found at https://stats.idre.ucla.edu/r/dae/logit-regression/

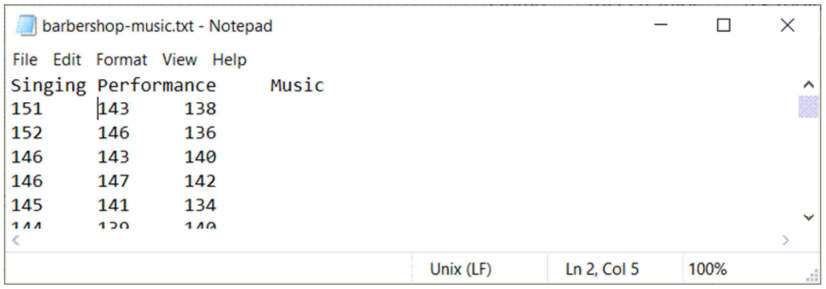
There is no formal data dictionary, which is a shame, but you can find a brief description on a second page where this data set is used.

Example #1

```
fn <-
"https://stats.idre.ucla.edu/stat/data/binary.csv
"
mydata <- read.csv(fn, stringsAsFactors=FALSE)
head(mydata)
##   admit gre  gpa rank
## 1     0 380 3.61    3
## 2     1 660 3.67    3
## 3     1 800 4.00    1
## 4     1 640 3.19    4
## 5     0 520 2.93    4
## 6     1 760 3.00    2
```

The read.csv file does a fine job here. I am only printing the first few rows of the data.

Example #2



This is a second dataset. Looking at it in notepad, it looks like it is a tab delimited file.

Example #2

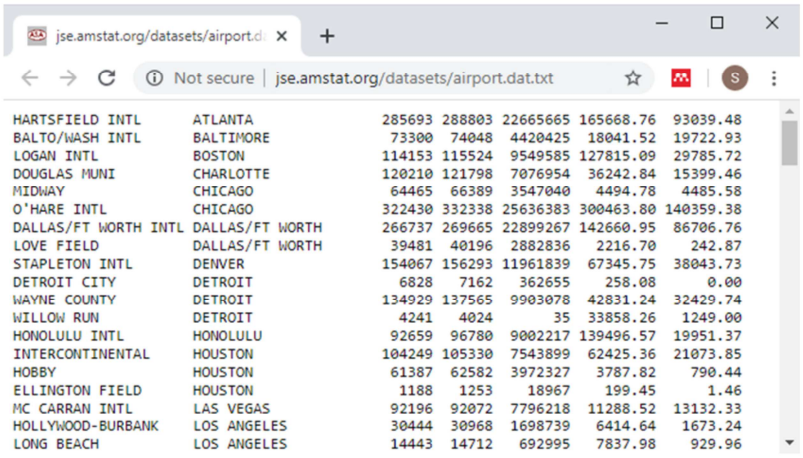
- No data dictionary
  - Brief description: "At a barbershop music singing competition, choruses are judged on three scales: Music (quality of the arrangement, etc.), Performance, and Singing."
  - Description found at https://dasl.datadescription.com/datafile/barbershop-music/

Example #2

```
fn <-
"https://dasl.datadescription.com/download/data/3
061"
mydata <- read.table(
  fn, header=TRUE, sep="\t",
  stringsAsFactors=FALSE)
head(mydata)
##   Singing Performance Music
## 1     151         143   138
## 2     152         146   136
## 3     146         143   140
## 4     146         147   142
## 5     145         141   134
## 6     144         139   140
```

The read.table command works well here. The separator or delimiter is the tab character, which is coded in R as

Example #3




```
HARTSFIELD INTL       ATLANTA          285693 288803 22665665 165668.76  93039.48
BALTO/WASH INTL       BALTIMORE         73300  74048  4420425  18041.52  19722.93
LOGAN INTL            BOSTON           114153 115524  9549585 127815.09  29785.72
DOUGLAS MUNI          CHARLOTTE        120210 121798  7076954  36242.84  15399.46
MIDWAY                CHICAGO           64465  66389  3547040   4494.78   4485.58
O'HARE INTL           CHICAGO          322430 332338 25636383 300463.80 140359.38
DALLAS/FT WORTH INTL  DALLAS/FT WORTH  266737 269665 22899267 142660.95  86706.76
LOVE FIELD            DALLAS/FT WORTH   39481  40196  2882836   2216.70    242.87
STAPLETON INTL        DENVER           154067 156293 11961839  67345.75  38043.73
DETROIT CITY          DETROIT            6828   7162   362655    258.08      0.00
WAYNE COUNTY          DETROIT          134929 137565  9903078  42831.24  32429.74
WILLOW RUN            DETROIT            4241   4024       35  33858.26   1249.00
HONOLULU INTL         HONOLULU          92659  96780  9002217 139496.57  19951.37
INTERCONTINENTAL      HOUSTON          104249 105330  7543899  62425.36  21073.85
HOBBY                 HOUSTON           61387  62582  3972327   3787.82    790.44
ELLINGTON FIELD       HOUSTON            1188   1253    18967    199.45      1.46
MC CARRAN INTL        LAS VEGAS         92196  92072  7796218  11288.52  13132.33
HOLLYWOOD-BURBANK     LOS ANGELES       30444  30968  1698739   6414.64   1673.24
LONG BEACH            LOS ANGELES       14443  14712   692995   7837.98    929.96
```

This is a third data set. Notice that there are no variable names at the top of the file.

Example #3

– Data dictionary at http://jse.amstat.org/datasets/airport.txt. Here is an excerpt.

```
VARIABLE DESCRIPTIONS:
Airport                          Columns 1-21
City                             Columns 22-43
Scheduled departures             Columns 44-49
Performed departures             Columns 51-56
Enplaned passengers              Columns 58-65
Enplaned revenue tons of freight Columns 67-75
Enplaned revenue tons of mail    Columns 77-85
```

This dataset has a very nice data dictionary. From the description, you can tell that it is a fixed width format.

Example #3

```
fn <-
"http://jse.amstat.org/datasets/airport.dat.txt"
raw_data <- read.fwf(fn,
  c(21, 22, 6, 7, 9, 10, 10), header=FALSE)
head(raw_data, 2)
##                      V1
## 1 HARTSFIELD INTL
## 2 BALTO/WASH INTL
##                       V2     V3     V4
## 1 ATLANTA                285693 288803
## 2 BALTIMORE               73300  74048
##         V5        V6       V7
## 1 22665665 165668.76 93039.48
## 2  4420425  18041.52 19722.93
```

You have to do a bit of math here. If the first variable ends in column 21 and the second variable ends in column 43, then the number of columns for the second variable is 43-21.

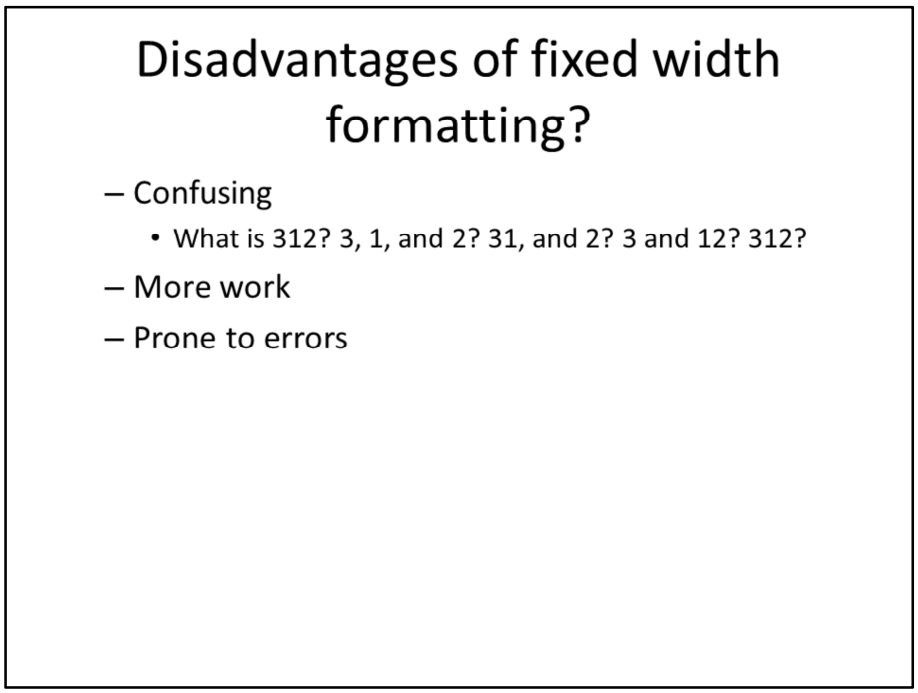
Should I download before reading?

- Read directly from website
  - Convenient
  - Updates incorporated at each run
- Download then read
  - Downloaded file doesn't disappear
  - Avoid repeated long downloads
  - Work even when Internet connection is down

R gives you the option of reading a file on your computer or reading it from a website. For small datasets that you only use once (such as for your homework assignments), it doesn't matter. For larger files and repeated data analyses, there are some advantages to reading directly from the website and some advantages to downloading the file to your computer.

Reading directly from the website is convenient. You don't have to figure out where to store your downloaded file. If the website updates the file on a regular basis, reading directly always insures that you have the most current data.

If you download the file and then read it, you provide yourself with some insurance against the website disappearing. If the download takes a long time, then you only have to endure that delay once. Finally, a downloaded file allows you to work when an Internet connection is not available, such as during a plane flight (though many airlines will now let you connect while in the air).

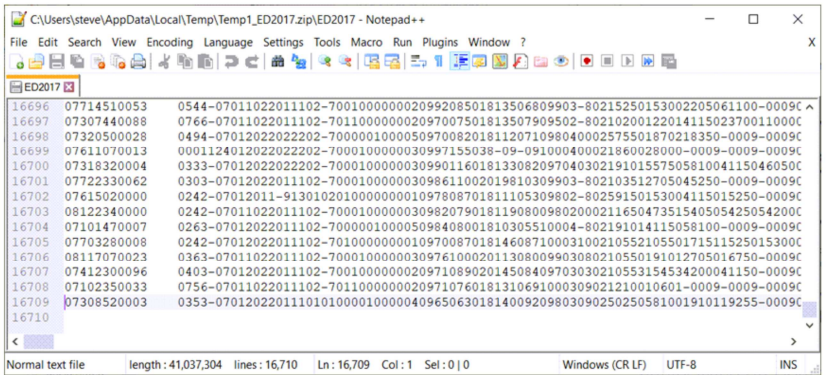


When you have no delimiters, it is easy to get confused. The third line, for example, of our simple data file is “312”. That could be three numbers: 3, 1, and 2. It could be two numbers: 31 and 2? Or maybe 3 and 12? Or it could be a single number: 312. In a small dataset, you won’t get confused, but you might with a larger dataset.

It’s also more work because you have to specify the number of columns that each variable uses. That’s not trivial for a large file.

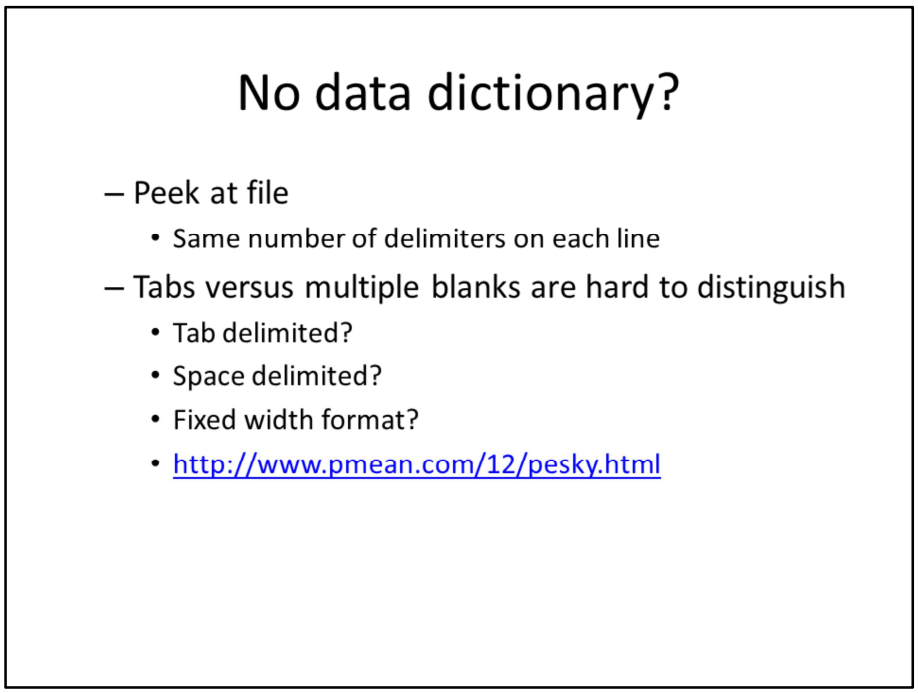
The fixed width format is also more prone to errors. If you get the columns wrong, you might truncate some of your data.

Example where fixed width formatting is needed.



Here's practical example where you really might need fixed width formatting. This is data from Emergency Department visits in a CDC survey. There are over 16,000 rows, which is bad enough. But each row (except the last one) has over 2,400 columns. These columns contain information for over 900 variables.

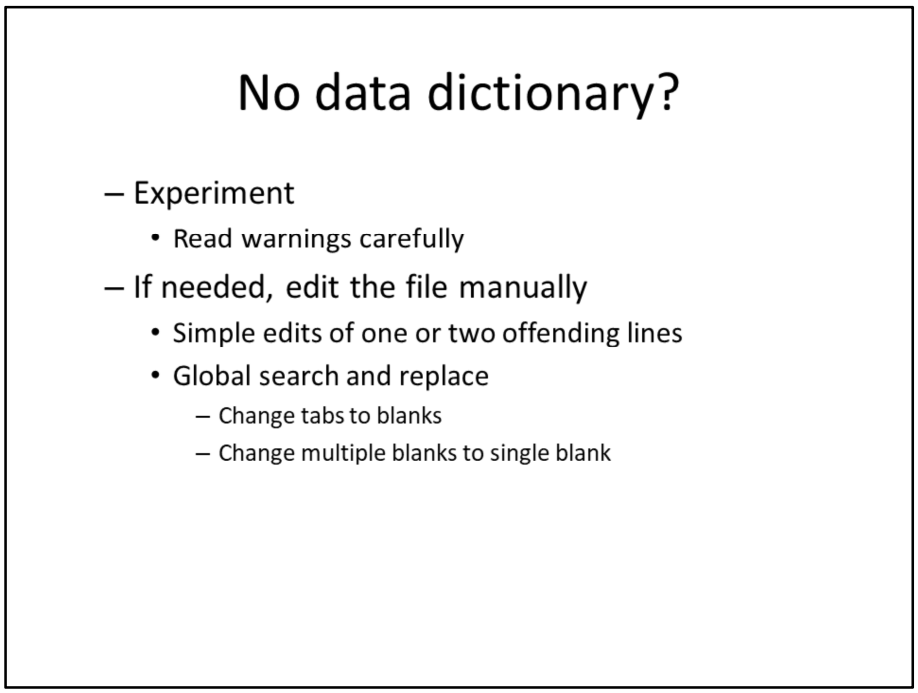
If you were to use delimiters in this file, you'd have to add 900 commas or 900 tabs or 900 spaces or 900 tildes to each and every line. That works out to be 1.4 million commas across the entire file. That's a substantial increase to the size of an already very large and unwieldy file.



Many files that you find on the Internet are missing any documentation or the documentation fails to help you figure out what approach to use to import the data. If that's the case, don't despair. There are several common sense things you can try.

First, peek at the file and see if there are any obvious delimiters. It's important that you have the exact same number of delimiters on each line of data. If you think the comma is the delimiter, then if there are five commas on one line, then every line should have five commas. The exception would be delimiters found inside quotes.

Tabs versus multiple blanks are hard to distinguish. This sometimes means that you will have difficulty telling whether to use a tab delimited file, a space delimited file, or a fixed width file. I have a web page that talks about this in detail.



There's nothing wrong with experimenting. Just pick one approach and try it. If you get an error message or the data is garbled, try a different approach. If you get warnings rather than errors, things may be okay, but look carefully at the warning message and double your vigilance efforts before you start analysis.

If all else fails, go in and edit the file manually. This helps if there are just a couple of rows in the file that are causing you heartburn. Usually the error or warning message will give you enough of a hint that you locate the offending lines. If there are problems on every line or almost every line, then sometimes a global search and replace works well.

Summary

- read.csv for comma delimited files
- read.table for other delimiters
  - Beware the tab
- read.fwf for fixed width files
- write with write.csv, write.table

Read data from text files is an important survival skill in R. Use the read.csv function for comma-delimited files. This is the most common format for text files.

Use read.table for other delimiters. Watch out for tab delimited files.

Use read.fwf for fixed width files.

If you need to write text files, the write.csv and write.table functions provide options similar to the two read functions.