Data importation woes

Matthew Castelo

June, 2021

This is an introduction to importing datasets into R, and some immediate problems that often present themselves. We will cover importing .csv files, Excel spreadsheets, and SAS datasets. In terms of data cleaning, we will also address quickly fixing non-standard variable names, fixing missing data indicators, and changing variable types (i.e. character to numeric).

We will need to install the following packages.

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

As usual, we will load our packages and set the working directory.

```r
library(tidyverse)
library(readxl) #This allows importing of Excel files
library(haven) #This allows importing of SAS files

#The working directory saves the following path when loading files, saving plots, etc
setwd("C:/Users/matth/Documents/R/R code and education/Tutorials/")
```

Importing .csv files

A .csv, or comma separated value file is a common format for datasets. Each line is an observation and values (columns) are separated by a comma. These files will often open in your standard spreadsheet software (i.e. Excel) and it might not be apparent at first they are different file types. CSV files do not have sheets like Excel files, and do not support formulae.

First, some history. For a long time, the base R function `read.csv()` had a very problematic default behaviour. Any variable that contained text would be automatically imported as a factor. Factors are categorical variables that have an explicitly defined order, or levels, to them. They allow you to convert a numeric variable like year of diagnosis to a categorical variable. It can also be useful to convert character variables to factors to control the order of output in legends, etc. However, routinely converting character strings to factors during data importation caused unexpected errors and made it difficult to reproduce analyses. Therefore, it was standard to include the argument `stringAsFactors = FALSE` when using `read.csv()`.

This annoying problem was addressed with the **Tidyverse** alternative, `read_csv()`, which by default imports strings as character variables. Note the underscore instead of the period. This function also conveniently prints the type of each variable after they've been imported. These days, even the default of `read.csv()` has been changed but I still prefer `read_csv()` as it meshes better with the rest of **Tidyverse**.

Our first dataset is on GitHub, so let's import it using this function.

```
urlRemote  <- "https://raw.githubusercontent.com/"
pathGithub <- "mcas-surg/Tutorials/main/Datasets/"
fileName   <- "generic_cancer.csv"

cancer <- read_csv(paste0(urlRemote, pathGithub, fileName))
```

Normally you will be importing files from your computer. The following format will make more sense.

```
cancer <- read_csv(
  "C:/Users/matth/Documents/R/R code and education/Tutorials/Datasets/generic_cancer.csv")

#As I have already set my working directory up to /Tutorials/ in this path,
#I can omit most of this pathway. The following is equivalent

cancer <- read_csv("Datasets/generic_cancer.csv")
```

You should see the type of each column print in the console. Let's take a closer look using `glimpse()` and `head()`.

```
glimpse(cancer)
```

```
## Rows: 80
## Columns: 13
## $ male                    <chr> "Male", "Male", "Female", "Male", "Male", "Male"~
## $ pt_id                   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1~
## $ age                     <dbl> 62, 57, 66, 54, 49, 63, 54, 72, 57, 65, 63, 54, ~
## $ high_grade              <chr> "No", "No", "No", "No", "No", "No", "No", "No", ~
## $ adverse_tumour_marker <chr> "No", "No", "No", "No", "No", "Yes", "No", "No",~
## $ tumour_size             <dbl> 0.41, 2.84, 3.35, 4.63, 1.29, 1.99, 1.86, 1.01, ~
## $ diabetes                <chr> "No", "No", "No", "No", "No", "No", "No", "No", ~
## $ heart_disease           <chr> "No", "No", "No", "Yes", "No", "No", "No", "No",~
## $ hospital                <chr> "Community hospital", "Community hospital", "Com~
## $ extended_resection      <chr> "Yes", "No", "No", "No", "No", "Yes", "No", "No"~
## $ postop_complication     <chr> "Yes", "No", "Yes", "Yes", "No", "Yes", "Yes", "~
## $ vital_status            <chr> "Alive", "Died", "Died", "Alive", "Alive", "Aliv~
## $ follow_up               <dbl> 58, 52, 33, 58, 56, 41, 64, 32, 30, 40, 29, 29, ~
```

```
head(cancer)
```

```
## # A tibble: 6 x 13
##   male   pt_id   age high_grade adverse_tumour_marker tumour_size diabetes
##   <chr>  <dbl> <dbl> <chr>      <chr>                       <dbl> <chr>
## 1 Male       1    62 No         No                           0.41 No
## 2 Male       2    57 No         No                           2.84 No
## 3 Female     3    66 No         No                           3.35 No
## 4 Male       4    54 No         No                           4.63 No
## 5 Male       5    49 No         No                           1.29 No
## 6 Male       6    63 No         Yes                          1.99 No
## # ... with 6 more variables: heart_disease <chr>, hospital <chr>,
## #   extended_resection <chr>, postop_complication <chr>, vital_status <chr>,
## #   follow_up <dbl>
```

To get a spreadsheet-type view, we can use.

```
View(cancer)
```

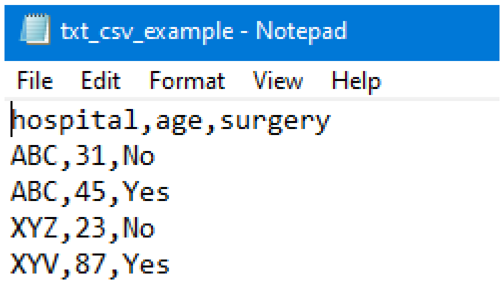
We can see the variables *pt_id*, *age*, *tumour_size*, and *follow_up* were imported as `<dbl>`, which is a type of numeric variable. Everything else was imported as a character.

The variable names also look good. R has some rules for variable names. They must start with a letter and can contain letters, numbers, periods, and underscores. R is case-sensitive. If the variable is called *Height*, trying to print `dataset$height` will not work. Variable names cannot have spaces in them.

This is an ideal case, where the dataset was provided as a .csv file and all the variable names are compatible. We will explore how to do some initial cleaning later in the tutorial.

Importing .txt files and other delimeters

Sometimes, a dataset will be CSV, i.e. values separated by commas, but they will not be in a .csv format. They will simply be pasted in a text file.



```
hospital,age,surgery
ABC,31,No
ABC,45,Yes
XYZ,23,No
XYV,87,Yes
```

This is treated the same as before using `read_csv()`, just make sure we indicate the correct file type as `file_name.txt`.

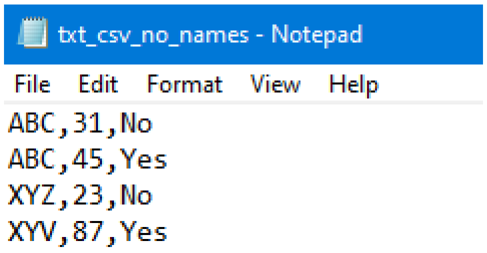
```
urlRemote  <- "https://raw.githubusercontent.com/"
pathGithub <- "mcas-surg/Tutorials/main/Datasets/"
fileName   <- "txt_csv_example.txt" #Here you can see the file name ends in .txt now

csv_as_text <- read_csv(paste0(urlRemote, pathGithub, fileName))

head(csv_as_text)
```

```
## # A tibble: 4 x 3
##   hospital   age surgery
##   <chr>    <dbl> <chr>
## 1 ABC         31 No
## 2 ABC         45 Yes
## 3 XYZ         23 No
## 4 XYV         87 Yes
```

Remember we are working from GitHub so you can download the files. On your computer this would look something like.

```
csv_as_text <- read_csv("Datasets/txt_csv_example.txt")
```

By default the first line will be treated as the column names. If they are not included, we must tell R to ignore the first line and we can manually provide the column names afterwards.



```
ABC,31,No
ABC,45,Yes
XYZ,23,No
XYV,87,Yes
```

If we import as normal.

```
urlRemote  <- "https://raw.githubusercontent.com/"
pathGithub <- "mcas-surg/Tutorials/main/Datasets/"
fileName   <- "txt_csv_no_names.txt"

csv_no_names <- read_csv(paste0(urlRemote, pathGithub, fileName))

head(csv_no_names)
```

```
## # A tibble: 3 x 3
##   ABC    `31` No
##   <chr> <dbl> <chr>
## 1 ABC      45 Yes
## 2 XYZ      23 No
## 3 XYV      87 Yes
```

The first line of data was imported as the column names. Let's set `col_names = FALSE`.

```
urlRemote  <- "https://raw.githubusercontent.com/"
pathGithub <- "mcas-surg/Tutorials/main/Datasets/"
fileName   <- "txt_csv_no_names.txt"

csv_no_names <- read_csv(paste0(urlRemote, pathGithub, fileName),
                         col_names = FALSE)

head(csv_no_names)
```

```
## # A tibble: 4 x 3
##   X1       X2 X3
##   <chr> <dbl> <chr>
## 1 ABC      31 No
## 2 ABC      45 Yes
## 3 XYZ      23 No
## 4 XYV      87 Yes
```

R automatically filled in the variables names with *X1*, *X2*, etc. We can name them whatever we want using the `colnames()` function. Normally, this prints out a vector of the column names.

```
colnames(csv_no_names)
```

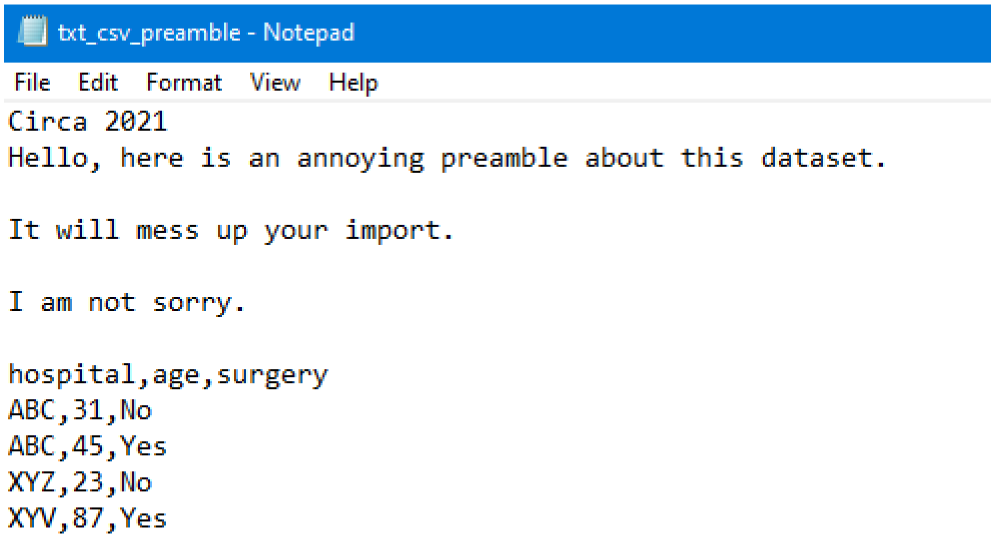
```
## [1] "X1" "X2" "X3"
```

If we pass a different vector to this they will be updated.

```
colnames(csv_no_names) <- c("hospital",
                            "age",
                            "surgery")

head(csv_no_names)
```

```
## # A tibble: 4 x 3
##   hospital    age surgery
##   <chr>     <dbl> <chr>
## 1 ABC          31 No
## 2 ABC          45 Yes
## 3 XYZ          23 No
## 4 XYV          87 Yes
```

Many of these .txt files will have a paragraph preceding the data giving some information about the dataset. This can be a problem during importation.



```
Circa 2021
Hello, here is an annoying preamble about this dataset.

It will mess up your import.

I am not sorry.

hospital,age,surgery
ABC,31,No
ABC,45,Yes
XYZ,23,No
XYV,87,Yes
```

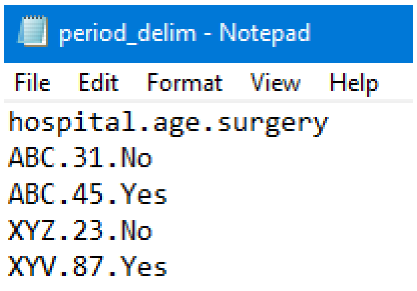
We can see there are 7 lines of nonsense. Let's skip these lines using the argument `skip = 7`.

```
urlRemote  <- "https://raw.githubusercontent.com/"
pathGithub <- "mcas-surg/Tutorials/main/Datasets/"
fileName   <- "txt_csv_preamble.txt"

csv_preamble <- read_csv(paste0(urlRemote, pathGithub, fileName),
                         skip = 7)

head(csv_preamble)
```

```
## # A tibble: 4 x 3
##   hospital   age surgery
##   <chr>    <dbl> <chr>
## 1 ABC         31 No
## 2 ABC         45 Yes
## 3 XYZ         23 No
## 4 XYV         87 Yes
```

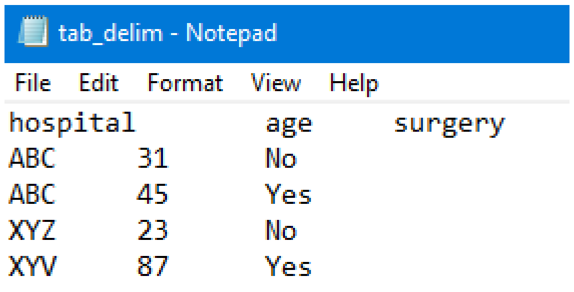
A CSV file is really one type of a larger group of data formats, called delimited files. A delimiter is a character that separates values. The delimiter in CSV files are, unsurprisingly, commas. You can have other delimiters such as periods, tabs, or backslashes. They can be read using the more flexible function `read_delim()` and telling R the specific delimiter.

Let's read in a period delimited and tab delimited file.



```
hospital.age.surgery
ABC.31.No
ABC.45.Yes
XYZ.23.No
XYV.87.Yes
```

```
urlRemote  <- "https://raw.githubusercontent.com/"
pathGithub <- "mcas-surg/Tutorials/main/Datasets/"
fileName   <- "period_delim.txt"

period_delim <- read_delim(paste0(urlRemote, pathGithub, fileName),
                           delim = ".")
```



```
hospital        age     surgery
ABC     31      No
ABC     45      Yes
XYZ     23      No
XYV     87      Yes
```

```
urlRemote  <- "https://raw.githubusercontent.com/"
pathGithub <- "mcas-surg/Tutorials/main/Datasets/"
fileName   <- "tab_delim.txt"

tab_delim <- read_delim(paste0(urlRemote, pathGithub, fileName),
                        delim = "\t")
```

Importing Excel files

Excel files have characteristics that can make them difficult to import, including data spread over sheets, inconsistent column structures, and blank rows/columns. We will use the **readxl** package and `read_excel()` function to import these files.

Excel files cannot be imported directly from GitHub, so you'll need to download the file from the test datasets: https://github.com/mcas-surg/Tutorials/blob/main/Datasets/excel_import_example.xlsx

Our file has three sheets, the first is the simplest case.

```
library(readxl)

simple_excel <- read_excel("Datasets/excel_import_example.xlsx",
                           sheet = "Sheet1")

head(simple_excel)
```

```
## # A tibble: 5 x 4
##   pt_id   age surgery outcome
##   <dbl> <dbl> <chr>   <chr>
## 1     1    32 Yes     Survived
## 2     2    45 Yes     Survived
## 3     3    67 No      Died
## 4     4    84 No      Died
## 5     5    23 Yes     Survived
```

Note we can specify the exact name of the sheet. If we don't it will default to the first sheet in the Excel file. In the second sheet of our example, the data are offset and we need to define an explicit range of the cells.

```r
offset_excel <- read_excel("Datasets/excel_import_example.xlsx",
                           sheet = "Sheet2",
                           range = "C5:F10")

head(offset_excel)
```

```
## # A tibble: 5 x 4
##   pt_id   age surgery outcome
##   <dbl> <dbl> <chr>   <chr>
## 1     1    32 Yes     Survived
## 2     2    45 Yes     Survived
## 3     3    67 No      Died
## 4     4    84 No      Died
## 5     5    23 Yes     Survived
```

Importing SAS files

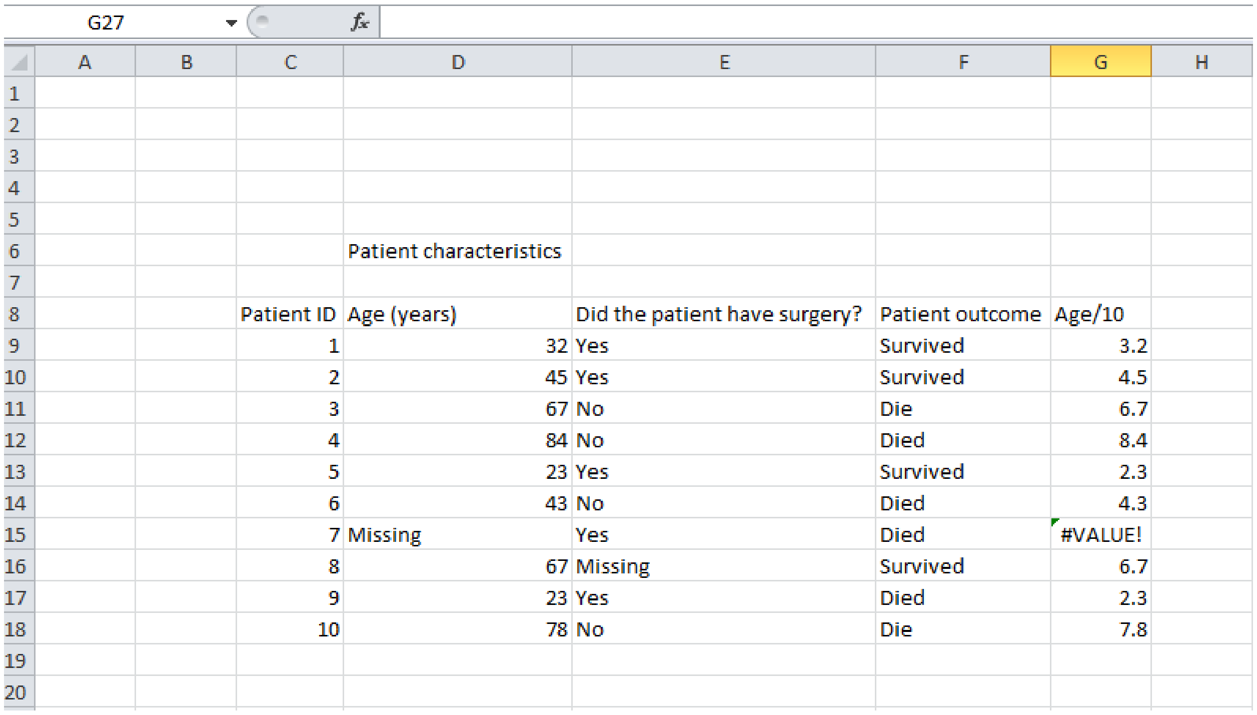
The **haven** package makes importing .sas7bdat files very straightforward. The `read_sas()` function works in much the same way as `read_csv()`. We can import an example taken from a dataset on primary biliary cirrhosis. The source is 'https://vincentarelbundock.github.io/Rdatasets/datasets.html' under the dataset name 'Mayo Clinic Primary Biliary Cirrhosis Data'.

```r
library(haven)

urlRemote  <- "https://raw.githubusercontent.com/"
pathGithub <- "mcas-surg/Tutorials/main/Datasets/"
fileName   <- "pbc.sas7bdat"

pbc <- read_sas(paste0(urlRemote, pathGithub, fileName))

head(pbc)
```

```
## # A tibble: 6 x 21
##   VAR1     id  time status trt     age sex   ascites hepato spiders edema  bili
##   <chr> <dbl> <dbl>  <dbl> <chr> <dbl> <chr> <chr>   <chr>  <chr>   <dbl> <dbl>
## 1 1         1   400      2 1      58.8 f     1       1      1         1    14.5
## 2 2         2  4500      0 1      56.4 f     0       1      1         0     1.1
## 3 3         3  1012      2 1      70.1 m     0       0      0         0.5   1.4
## 4 4         4  1925      2 1      54.7 f     0       1      1         0.5   1.8
## 5 5         5  1504      1 2      38.1 f     0       1      1         0     3.4
## 6 6         6  2503      2 2      66.3 f     0       1      0         0     0.8
## # ... with 9 more variables: chol <chr>, albumin <dbl>, copper <chr>,
## #   alk.phos <chr>, ast <chr>, trig <chr>, platelet <chr>, protime <chr>,
## #   stage <chr>
```

Common cleaning issues

Now that we've covered importing the most common file types, we can review issues that come up with importing. Let's use the third sheet of our Excel file, which has the following form.


G27 fx

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6				Patient characteristics				
7								
8			Patient ID	Age (years)	Did the patient have surgery?	Patient outcome	Age/10	
9			1	32	Yes	Survived	3.2	
10			2	45	Yes	Survived	4.5	
11			3	67	No	Die	6.7	
12			4	84	No	Died	8.4	
13			5	23	Yes	Survived	2.3	
14			6	43	No	Died	4.3	
15			7	Missing	Yes	Died	#VALUE!	
16			8	67	Missing	Survived	6.7	
17			9	23	Yes	Died	2.3	
18			10	78	No	Die	7.8	
19								
20								

Here are issues we can identify

1. Excel file with the data on the third sheet
2. Need to set the correct range of the data
3. The non-estimable formula giving the "#VALUE!" error
4. Missing data specified by the text "Missing"
5. Non-compatible variable names
6. Multiple text strings indicating patient died

The first two issues can be solved with the previous options we learned during importation.

```r
complex_excel <- read_excel("Datasets/excel_import_example.xlsx",
                            sheet = "Sheet3",
                            range = "C8:G18")

complex_excel
```

```
## # A tibble: 10 x 5
##    `Patient ID` `Age (years)` `Did the patient have s~ `Patient outcom~ `Age/10`
##           <dbl> <chr>         <chr>                    <chr>               <dbl>
##  1            1 32            Yes                      Survived              3.2
##  2            2 45            Yes                      Survived              4.5
##  3            3 67            No                       Die                   6.7
##  4            4 84            No                       Died                  8.4
##  5            5 23            Yes                      Survived              2.3
##  6            6 43            No                       Died                  4.3
##  7            7 Missing       Yes                      Died                 NA  
##  8            8 67            Missing                  Survived              6.7
##  9            9 23            Yes                      Died                  2.3
## 10           10 78            No                       Die                   7.8
```

Note the `NA` value in the *Age/10* column. R already recognized the formula error in Excel and made the value missing. We don't need to do anything else, so the first three issues are solved.

Missing data has been specified by the text "Missing". This can be fixed during import by setting the `na =` option. Let's update our import program.

```
complex_excel <- read_excel("Datasets/excel_import_example.xlsx",
                            sheet = "Sheet3",
                            range = "C8:G18",
                            na = "Missing")

complex_excel
```

```
## # A tibble: 10 x 5
##    `Patient ID` `Age (years)` `Did the patient have s~ `Patient outcom~ `Age/10`
##           <dbl>         <dbl> <chr>                    <chr>               <dbl>
##  1            1            32 Yes                      Survived              3.2
##  2            2            45 Yes                      Survived              4.5
##  3            3            67 No                       Die                   6.7
##  4            4            84 No                       Died                  8.4
##  5            5            23 Yes                      Survived              2.3
##  6            6            43 No                       Died                  4.3
##  7            7            NA Yes                      Died                NA
##  8            8            67 <NA>                     Survived              6.7
##  9            9            23 Yes                      Died                  2.3
## 10           10            78 No                       Die                   7.8
```

If you look back, you will notice this also fixed *Age (years)* being imported as a character variable due to the "Missing" string. It is now correctly a numeric variable.

Unfortunately, the variable names are not R-compatible. We can see they are wrapped in single quotes. When we have non-compatible variable names we can still refer to them by using this strategy. For example, selecting the *Patient ID* variable, which contains a space.

```
complex_excel %>%
  select(`Patient ID`)
```

```
## # A tibble: 10 x 1
##    `Patient ID`
##           <dbl>
##  1            1
##  2            2
##  3            3
##  4            4
##  5            5
##  6            6
##  7            7
##  8            8
##  9            9
## 10           10
```

Working with non-standard variable names is obviously more work and will cause errors. Let's use the `colnames()` strategy and change them.

```
colnames(complex_excel) <- c("patient_id",
                             "age",
                             "surgery",
                             "outcome",
                             "age_divided")

complex_excel
```

```
## # A tibble: 10 x 5
##    patient_id   age surgery outcome  age_divided
##         <dbl> <dbl> <chr>   <chr>          <dbl>
##  1          1    32 Yes     Survived         3.2
##  2          2    45 Yes     Survived         4.5
##  3          3    67 No      Die              6.7
##  4          4    84 No      Died             8.4
##  5          5    23 Yes     Survived         2.3
##  6          6    43 No      Died             4.3
##  7          7    NA Yes     Died            NA  
##  8          8    67 <NA>    Survived         6.7
##  9          9    23 Yes     Died             2.3
## 10         10    78 No      Die              7.8
```

The final issue is the *outcome* variable. We have values for both "Die" and "Died". We need to create a common value. We can accomplish this and convert the variable to a factor in the same step. The `factor()` function has two important arguments - `levels =` indicates the desired order the categories, and `labels =` lets you change the values themselves, including combining categories.

Let's convert *outcome* to a factor.

```
complex_excel <- complex_excel %>%
  mutate(outcome = factor(outcome,
                          levels = c("Survived", "Died", "Die"),
                          labels = c("Survived", "Died", "Died")))

complex_excel
```

```
## # A tibble: 10 x 5
##    patient_id   age surgery outcome  age_divided
##         <dbl> <dbl> <chr>   <fct>          <dbl>
##  1          1    32 Yes     Survived         3.2
##  2          2    45 Yes     Survived         4.5
##  3          3    67 No      Died             6.7
##  4          4    84 No      Died             8.4
##  5          5    23 Yes     Survived         2.3
##  6          6    43 No      Died             4.3
##  7          7    NA Yes     Died            NA  
##  8          8    67 <NA>    Survived         6.7
##  9          9    23 Yes     Died             2.3
## 10         10    78 No      Died             7.8
```

We can show summary statistics to look for final problems.

```
summary(complex_excel)
```

```
##   patient_id          age           surgery               outcome  
##  Min.   : 1.00   Min.   :23.00   Length:10          Survived:4  
##  1st Qu.: 3.25   1st Qu.:32.00   Class :character   Died    :6  
##  Median : 5.50   Median :45.00   Mode  :character               
##  Mean   : 5.50   Mean   :51.33                                  
##  3rd Qu.: 7.75   3rd Qu.:67.00                                  
##  Max.   :10.00   Max.   :84.00                                  
##                  NA's   :1                                      
##   age_divided   
##  Min.   :2.300  
##  1st Qu.:3.200  
##  Median :4.500  
##  Mean   :5.133  
##  3rd Qu.:6.700  
##  Max.   :8.400  
##  NA's   :1
```

Since we never changed *surgery* to a factor it cannot be summarized in this method. We will also change it to a factor.

```
complex_excel <- complex_excel %>%
  mutate(surgery = factor(surgery))

summary(complex_excel)
```

```
##   patient_id          age          surgery       outcome    age_divided   
##  Min.   : 1.00   Min.   :23.00   No  :4    Survived:4   Min.   :2.300  
##  1st Qu.: 3.25   1st Qu.:32.00   Yes :5    Died    :6   1st Qu.:3.200  
##  Median : 5.50   Median :45.00   NA's:1                 Median :4.500  
##  Mean   : 5.50   Mean   :51.33                          Mean   :5.133  
##  3rd Qu.: 7.75   3rd Qu.:67.00                          3rd Qu.:6.700  
##  Max.   :10.00   Max.   :84.00                          Max.   :8.400  
##                  NA's   :1                              NA's   :1
```

Now that our dataset is cleaned, a quick check can be done for missing data. The function `is.na()` will provide `TRUE` or `FALSE` for each value depending on its missing status. For example, on a single variable.

```
is.na(complex_excel$surgery)
```

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

We see one `TRUE`. This is hard to read, so we can ask R to sum the column. `TRUE` is treated as 1 and `FALSE` as 0 by default.

```
sum(is.na(complex_excel$surgery))
```

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

As expected, 1 missing. We can extend this to all the columns using `colSums()` and passing the entire dataset to `is.na()`.

```
colSums(is.na(complex_excel))
```

```
##  patient_id         age     surgery     outcome age_divided
##           0           1           1           0           1
```

To summarize, we have learned how to import .csv files, general text delimited files, Excel files, and SAS files. We have also gone over a short example of data cleaning.