LeaRning Week 2

LeaRning Team

27-May-2021

Introduction

```
##
##  -----
## Welcome to LeaRning: Session #2!
##  ------
##     \
##      \
##                  \`*-.
##                   )  _`-.
##                  .  : `. .
##                  : _   '  \
##                  ; *` _.   `*-._
##                  `-.-'          `-.
##                    ;       `       `.
##                    :.       .        \
##                    .\  .   :   .-'   .
##                    '  `+.;  ;  '      :
##                    :  '  |    ;       ;-.
##                    ; '   : :`-:     _.`* ;
##                 .*' /  .*' ; .*`- +'  `*'
##      [bug]      `*-*   `*-*  `*-*'
##
```

Welcome to the second session for LeaRning! This lesson will equip you with the skills to:

1. Install and load packages within R - this will ensure you can make the most of the functionality available
2. Load your data into R
3. Work with data frames - one of the most common R data types!
4. Export your files back to the working directory

Part 1: Install and load R packages

To install a package, one of the easiest ways is to use the function `install.packages()`

Here, we will use this function to install the package **readxl**

```
install.packages("readxl")
```

Fantastic! We now have access to all of the functionality contained within the **readxl** package. To start using this package, we first need to "load" the library i.e. connect our current RStudio session with this package using the `library()` function.

```
library("readxl")

# note that the quotation marks here are optional, this function also works without them

library(readxl)
```

You only need to **install** a package once, but you must **load** the package in each new RStudio session.

Another handy trick is that you can check if you already have a package installed using the following code:

```
# lets double check that the readxl package is installed
any(grepl("readxl",
          installed.packages()))
```

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

Part 2: Loading data

2.1 Preparing your data

There are a few different ways to load your data into R, depending on the format. We commonly store data in Excel spreadsheets, CSV (comma separated values) files or delimited text files. To make your life easier when reading data into R there are a few simple things you can do first:

1. Ensure that the first row is reserved for the header (descriptions of columns) and the first column contains sample IDs

2. Avoid using blank spaces for column descriptions, use "." or "_" instead to separate words

3. Keep column descriptions short and try to avoid symbols like ?, $, %, ^, &, *, (,), -, #, ?, <, >, /, |, , [,], {, and }

4. Delete any comments you have made in your Excel file - this may introduce NAs when reading data into R

5. Make sure any missing values are indicated with NA

6. Save your Excel file as a .csv or .txt file

2.2 Loading data into R

Now that our data is tidy, we are ready to load it into R. First, it's a good idea to check where our current working directory is set to - we need to make sure that this is the same place that our data files are stored:

```
# use the getwd() function to print the file path of our current working directory
getwd()
```

```
## [1] "O:/EmmaDeJong/LeaRning"
```

If we need to change the current working directory, we can use the `setwd()` command:

```
setwd("<place your file path here>")

# for example
```

```
setwd("C:/Users/your_username/Desktop/LeaRning")
```

Now, to read in the data we have two options: 1) To use basic R commands (already built into R) or 2) to use packages specifically designed for reading in certain file types.

Let's look at the basic R commands first, which focus on reading in things like Excel spreadsheets saved in other formats (.txt or .csv) rather than the actual Excel files. The `read.table()` function is one of the most common and most simple ways to import your file into R:

```
# use the help documentation to see what the arguments of the read.table() function are,
# and their default values
?read.table()

# now read in the data, remember to assign your data to an object  - otherwise what will happen?
df <- read.table(file = "LeaRning_week2_cfu_data.txt")
```

Let's take a look at the data we have just read into RStudio:

```
# the head() function will show us the top 6 rows of our data
head(df)
```

```
##          V1   V2              V3  V4                 V5     V6              V7
## 1 animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 2        R2   G7      2016.01.12   F          Treatment  liver              10
## 3        R4   G7      2016.01.12   F          Treatment  liver              10
## 4        R1   G9      2016.01.13   M          Treatment  liver              10
## 5        R2   G9      2016.01.13   M          Treatment  liver              10
## 6        R3   G9      2016.01.13   F          Treatment  liver              10
##                   V8            V9           V10    V11
## 1 homogenate.volume plated.volume tissue.weight no.cfu
## 2                 3          0.04        0.1415     60
## 3                 3          0.04        0.0679      0
## 4                 3          0.04        0.0999     62
## 5                 3          0.04        0.0611      0
## 6                 3          0.04         0.075     47
```

What do you notice? What are V1, V2, V3 etc?

You may recall from the help documentation for `read.table()` that by default, the argument for "header" is set to FALSE. What does this actually mean? Well, by keeping header = FALSE, we are telling R that the top row just contains another row of data, not our column names. Let's fix this by setting header to TRUE:

```
df <- read.table(file = "LeaRning_week2_cfu_data.txt",
                 header = TRUE)

# again, let's have a look at the data
head(df)
```

```
##   animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 1        R2   G7      2016.01.12   F          Treatment  liver              10
## 2        R4   G7      2016.01.12   F          Treatment  liver              10
## 3        R1   G9      2016.01.13   M          Treatment  liver              10
## 4        R2   G9      2016.01.13   M          Treatment  liver              10
## 5        R3   G9      2016.01.13   F          Treatment  liver              10
## 6        F4   G9      2016.01.13   M            Control  liver              10
##   homogenate.volume plated.volume tissue.weight no.cfu
## 1                 3          0.04        0.1415     60
```

```
## 2                3          0.04        0.0679      0
## 3                3          0.04        0.0999     62
## 4                3          0.04        0.0611      0
## 5                3          0.04        0.0750     47
## 6                3          0.04        0.0294      0
```

Perfect! We now have our data loaded into R, and saved as the object "df" ready for analysis! But what if we have a .csv file instead of a .txt file? Easy! We can use the `read.csv()` function:

```
?read.csv()

df <- read.csv(file = "LeaRning_week2_cfu_data.csv")

# let's take a look at the data
head(df)
```

```
##   animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 1        R2   G7      2016.01.12   F          Treatment  liver              10
## 2        R4   G7      2016.01.12   F          Treatment  liver              10
## 3        R1   G9      2016.01.13   M          Treatment  liver              10
## 4        R2   G9      2016.01.13   M          Treatment  liver              10
## 5        R3   G9      2016.01.13   F          Treatment  liver              10
## 6        F4   G9      2016.01.13   M            Control  liver              10
##   homogenate.volume plated.volume tissue.weight no.cfu
## 1                 3          0.04        0.1415     60
## 2                 3          0.04        0.0679      0
## 3                 3          0.04        0.0999     62
## 4                 3          0.04        0.0611      0
## 5                 3          0.04        0.0750     47
## 6                 3          0.04        0.0294      0
```

What do you notice?

One of the differences between `read.csv()` and `read.table()` is that by default, the header argument is set to TRUE for `read.csv()`. That means, we don't have to explicitly tell R that our column names are contained in the first row. Essentially, `read.csv()` is a variant of the `read.table()` function. Another important difference is that the default separator symbol in `read.csv()` is naturally, a comma - instead of white space for `read.table()`.

Technically, you can read in your .csv files using the `read.table()` function, by adjusting some of the function arguments (parameters):

```
df <- read.table(file = "LeaRning_week2_cfu_data.csv",
                 header = TRUE,
                 sep = ",")

head(df)
```

```
##   animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 1        R2   G7      2016.01.12   F          Treatment  liver              10
## 2        R4   G7      2016.01.12   F          Treatment  liver              10
## 3        R1   G9      2016.01.13   M          Treatment  liver              10
## 4        R2   G9      2016.01.13   M          Treatment  liver              10
## 5        R3   G9      2016.01.13   F          Treatment  liver              10
## 6        F4   G9      2016.01.13   M            Control  liver              10
##   homogenate.volume plated.volume tissue.weight no.cfu
## 1                 3          0.04        0.1415     60
## 2                 3          0.04        0.0679      0
```

```
## 3                 3          0.04        0.0999     62
## 4                 3          0.04        0.0611      0
## 5                 3          0.04        0.0750     47
## 6                 3          0.04        0.0294      0
```

So why would we bother with `read.csv()`? Well, it's just easier! And where possible, it's good practice to keep your code as clean and simple as possible.

Another variation of the `read.table()` function is `read.delim()`, which again differs in that header = TRUE (telling R that first line that is being read in is a header with the attribute names), and sep = "\t" which indicates that our values are separated by a tab.

Now let's take a look at how we can read in an Excel file using the **readxl** package we installed earlier:

```
# look at the help documentation for extra info
?readxl

# read in our Excel file
df <- read_excel("LeaRning_week2_cfu_data.xlsx")
```

Let's check the data:

```
head(df)
```

```
## # A tibble: 0 x 0
```

Weird.... what happened?

Turns out that the `read_excel()` assumes your data is in the first sheet. We can check how many sheets are in our Excel file using the `excel_sheets()` function:

```
excel_sheets("LeaRning_week2_cfu_data.xlsx")
```

```
## [1] "OtherNotes"              "LeaRning_week2_cfu_data"
```

OK so to properly read in our data, we need specify the name of the sheet:

```
df <- read_excel("LeaRning_week2_cfu_data.xlsx",
                 sheet = "LeaRning_week2_cfu_data")
```

```
head(df)
```

```
## # A tibble: 6 x 11
##   `CFU experiment:~ ...2  ...3   ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
##   <chr>             <chr> <chr>  <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 animal.id         cage  exper~ sex   expe~ tiss~ dilu~ homo~ plat~ tiss~ no.c~
## 2 R2                G7    2016.~ F     Trea~ liver 10    3     0.04  0.14~ 60
## 3 R4                G7    2016.~ F     Trea~ liver 10    3     0.04  6.79~ 0
## 4 R1                G9    2016.~ M     Trea~ liver 10    3     0.04  9.99~ 62
## 5 R2                G9    2016.~ M     Trea~ liver 10    3     0.04  6.11~ 0
## 6 R3                G9    2016.~ F     Trea~ liver 10    3     0.04  7.49~ 47
```

This also looks weird!

Remember how we have a title in our Excel worksheet? Well, R assumes that this is the header row containing column names. We can skip this row by setting "skip = 1":

```
df <- read_excel("LeaRning_week2_cfu_data.xlsx",
                 sheet = "LeaRning_week2_cfu_data",
                 skip = 1)

head(df)
```

```
## # A tibble: 6 x 11
##   animal.id cage  experiment.date sex   experimental.gro~ tissue dilution.factor
##   <chr>     <chr> <chr>           <chr> <chr>             <chr>            <dbl>
## 1 R2        G7    2016.01.12      F     Treatment         liver               10
## 2 R4        G7    2016.01.12      F     Treatment         liver               10
## 3 R1        G9    2016.01.13      M     Treatment         liver               10
## 4 R2        G9    2016.01.13      M     Treatment         liver               10
## 5 R3        G9    2016.01.13      F     Treatment         liver               10
## 6 F4        G9    2016.01.13      M     Control           liver               10
## # ... with 4 more variables: homogenate.volume <dbl>, plated.volume <dbl>,
## #   tissue.weight <chr>, no.cfu <dbl>
```

You will notice that this data looks slightly different to when we read it in using `read.table()` or `read.csv()`. That is because the **readxl** package converts the data into a different format (beyond the scope of this lesson). Remember, we can convert between data types. Let's try now to convert this data to a regular data frame:

```
df <- as.data.frame(df)
```

```
head(df)
```

```
##   animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 1        R2   G7      2016.01.12   F          Treatment  liver              10
## 2        R4   G7      2016.01.12   F          Treatment  liver              10
## 3        R1   G9      2016.01.13   M          Treatment  liver              10
## 4        R2   G9      2016.01.13   M          Treatment  liver              10
## 5        R3   G9      2016.01.13   F          Treatment  liver              10
## 6        F4   G9      2016.01.13   M            Control  liver              10
##   homogenate.volume plated.volume        tissue.weight no.cfu
## 1                 3          0.04   0.14149999999999999     60
## 2                 3          0.04 6.7900000000000002E-2      0
## 3                 3          0.04 9.9900000000000003E-2     62
## 4                 3          0.04 6.1100000000000002E-2      0
## 5                 3          0.04 7.4999999999999997E-2     47
## 6                 3          0.04 2.9399999999999999E-2      0
```

Perfect! You now have all the skills you need to get your data loaded into R, from a variety of formats. Next, we will practice skills in working with data frames.

Part 3: Working with data frames

Like we discussed in the first session, looking at our data in R may not be as straight forward as opening it up in Excel, but once you master a few simple tricks, you'll see how easy and efficiently you can understand your data in R! Below are some of the most common and handy functions we can use to get a good overview of our data.

3.1 Handy functions

Here is a list of some of the most useful in-built functions to check your data, which we will go through one by one:

1. `head()`
2. `tail()`
3. `dim()`

4. `str()`
5. `summary()`
6. `table()`
7. `unique()`

We have already used `head()` to print the first 6 rows of our data frames. But what if we want to see more rows? Easy! We can specify the number of rows like this:

```
head(df, n = 10) # this will print the first 10 rows
```

```
##    animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 1         R2   G7      2016.01.12   F          Treatment  liver              10
## 2         R4   G7      2016.01.12   F          Treatment  liver              10
## 3         R1   G9      2016.01.13   M          Treatment  liver              10
## 4         R2   G9      2016.01.13   M          Treatment  liver              10
## 5         R3   G9      2016.01.13   F          Treatment  liver              10
## 6         F4   G9      2016.01.13   M            Control  liver              10
## 7         R4   G9      2016.01.13   M          Treatment  liver              10
## 8         R1   G8      2016.01.16   M          Treatment  liver              10
## 9         R2   G8      2016.01.16   M          Treatment  liver              10
## 10        R3   G8      2016.01.16   M          Treatment  liver              10
##    homogenate.volume plated.volume         tissue.weight no.cfu
## 1                  3          0.04   0.14149999999999999     60
## 2                  3          0.04 6.7900000000000002E-2      0
## 3                  3          0.04 9.9900000000000003E-2     62
## 4                  3          0.04 6.1100000000000002E-2      0
## 5                  3          0.04 7.4999999999999997E-2     47
## 6                  3          0.04 2.9399999999999999E-2      0
## 7                  3          0.04 8.3099999999999993E-2     31
## 8                  3          0.04 7.0000000000000007E-2      0
## 9                  3          0.04 6.6900000000000001E-2     52
## 10                 3          0.04 6.8000000000000005E-2      0
```

As you may have guessed, `tail()` shows the *last* 6 rows of a data frame:

```
tail(df)
```

```
##    animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 91        F3   G8      2016.01.16   M            Control spleen            1000
## 92        R3   G9      2016.01.16   M            Control spleen            1000
## 93        R1   G9      2016.01.16   M            Control spleen           10000
## 94        R2   G9      2016.01.16   F            Control spleen           10000
## 95        R4   G9      2016.01.16   F            Control spleen           10000
## 96        R5   G9      2016.01.16   M            Control spleen           10000
##    homogenate.volume plated.volume         tissue.weight no.cfu
## 91               0.3          0.04 4.7999999999999996E-3     23
## 92               0.3          0.04 1.2999999999999999E-2     35
## 93               0.3          0.04 3.1099999999999999E-2     48
## 94               0.3          0.04               1.41E-2     48
## 95               0.3          0.04               1.24E-2     20
## 96               0.3          0.04               1.43E-2     29
```

Again, we can specify how many rows we want to look at:

```r
tail(df, n = 15)
```

```
##    animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 82        F4   G9      2016.01.13   M            Control spleen             100
## 83        R4   G9      2016.01.13   M          Treatment spleen             100
## 84        F1   G7      2016.01.12   F            Control spleen            1000
## 85        F2   G7      2016.01.12   M            Control spleen            1000
## 86        F3   G7      2016.01.12   F            Control spleen            1000
## 87        F4   G7      2016.01.12   M            Control spleen            1000
## 88        F3   G9      2016.01.13   F            Control spleen            1000
## 89        F1   G8      2016.01.16   M            Control spleen            1000
## 90        F2   G8      2016.01.16   F            Control spleen            1000
## 91        F3   G8      2016.01.16   M            Control spleen            1000
## 92        R3   G9      2016.01.16   M            Control spleen            1000
## 93        R1   G9      2016.01.16   M            Control spleen           10000
## 94        R2   G9      2016.01.16   F            Control spleen           10000
## 95        R4   G9      2016.01.16   F            Control spleen           10000
## 96        R5   G9      2016.01.16   M            Control spleen           10000
##    homogenate.volume plated.volume         tissue.weight no.cfu
## 82               0.3          0.04 4.7000000000000002E-3     12
## 83               0.3          0.04 5.5999999999999999E-3     14
## 84               0.3          0.04 8.6999999999999994E-3     25
## 85               0.3          0.04               1.21E-2     34
## 86               0.3          0.04               1.46E-2     63
## 87               0.3          0.04               2.64E-2     10
## 88               0.3          0.04                1.6E-2     19
## 89               0.3          0.04 1.5299999999999999E-2     33
## 90               0.3          0.04 3.2000000000000002E-3     42
## 91               0.3          0.04 4.7999999999999996E-3     23
## 92               0.3          0.04 1.2999999999999999E-2     35
## 93               0.3          0.04 3.1099999999999999E-2     48
## 94               0.3          0.04               1.41E-2     48
## 95               0.3          0.04               1.24E-2     20
## 96               0.3          0.04               1.43E-2     29
```

```r
# note, that we can omit the "n =" and just include a number
tail(df, 15)
```

```
##    animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 82        F4   G9      2016.01.13   M            Control spleen             100
## 83        R4   G9      2016.01.13   M          Treatment spleen             100
## 84        F1   G7      2016.01.12   F            Control spleen            1000
## 85        F2   G7      2016.01.12   M            Control spleen            1000
## 86        F3   G7      2016.01.12   F            Control spleen            1000
## 87        F4   G7      2016.01.12   M            Control spleen            1000
## 88        F3   G9      2016.01.13   F            Control spleen            1000
## 89        F1   G8      2016.01.16   M            Control spleen            1000
## 90        F2   G8      2016.01.16   F            Control spleen            1000
## 91        F3   G8      2016.01.16   M            Control spleen            1000
## 92        R3   G9      2016.01.16   M            Control spleen            1000
## 93        R1   G9      2016.01.16   M            Control spleen           10000
## 94        R2   G9      2016.01.16   F            Control spleen           10000
## 95        R4   G9      2016.01.16   F            Control spleen           10000
## 96        R5   G9      2016.01.16   M            Control spleen           10000
##    homogenate.volume plated.volume         tissue.weight no.cfu
```

```
## 82              0.3          0.04 4.7000000000000002E-3     12
## 83              0.3          0.04 5.5999999999999999E-3     14
## 84              0.3          0.04 8.6999999999999994E-3     25
## 85              0.3          0.04               1.21E-2     34
## 86              0.3          0.04               1.46E-2     63
## 87              0.3          0.04               2.64E-2     10
## 88              0.3          0.04                1.6E-2     19
## 89              0.3          0.04 1.5299999999999999E-2     33
## 90              0.3          0.04 3.2000000000000002E-3     42
## 91              0.3          0.04 4.7999999999999996E-3     23
## 92              0.3          0.04 1.2999999999999999E-2     35
## 93              0.3          0.04 3.1099999999999999E-2     48
## 94              0.3          0.04               1.41E-2     48
## 95              0.3          0.04               1.24E-2     20
## 96              0.3          0.04               1.43E-2     29
```

OK so we can easily see the top and bottom of our data frame, but how can we get a better idea of what the entire dataset looks like? We can use `dim()` to print the dimension of our data:

```
dim(df)
```

```
## [1] 96 11
```

The output tells us our data frame consists of 96 rows and 11 columns (remember that in R, we always talk about data frames in terms of rows, then columns).

We can also use the `str()` (structure) function to get additional information including the data types for each column, and a preview of the data:

```
str(df)
```

```
## 'data.frame':    96 obs. of  11 variables:
##  $ animal.id         : chr  "R2" "R4" "R1" "R2" ...
##  $ cage              : chr  "G7" "G7" "G9" "G9" ...
##  $ experiment.date   : chr  "2016.01.12" "2016.01.12" "2016.01.13" "2016.01.13" ...
##  $ sex               : chr  "F" "F" "M" "M" ...
##  $ experimental.group: chr  "Treatment" "Treatment" "Treatment" "Treatment" ...
##  $ tissue            : chr  "liver" "liver" "liver" "liver" ...
##  $ dilution.factor   : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ homogenate.volume : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ plated.volume     : num  0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 ...
##  $ tissue.weight     : chr  "0.14149999999999999" "6.7900000000000002E-2" "9.9900000000000003E-2" "6
##  $ no.cfu            : num  60 0 62 0 47 0 31 0 52 0 ...
```

What do we notice here? Are there any obvious changes we might want to make to this data set?

You can see that several variables are of type "chr" (character) where it probably makes more sense for these to be categorical, or type "factor". We will see how to do this in the next section.

Next, we can use the function `summary()` for a quick statistical breakdown of each column - most useful for the "num" (numeric) data types:

```
summary(df)
```

```
##   animal.id             cage           experiment.date        sex
##  Length:96          Length:96          Length:96          Length:96
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
```

```
##
##
##  experimental.group    tissue          dilution.factor   homogenate.volume
##  Length:96          Length:96          Min.   :   10.0   Min.   :0.3
##  Class :character   Class :character   1st Qu.:   10.0   1st Qu.:0.3
##  Mode  :character   Mode  :character   Median :   10.0   Median :3.0
##                                        Mean   :  743.1   Mean   :2.1
##                                        3rd Qu.:  325.0   3rd Qu.:3.0
##                                        Max.   :10000.0   Max.   :3.0
##  plated.volume  tissue.weight          no.cfu
##  Min.   :0.04   Length:96          Min.   : 0.00
##  1st Qu.:0.04   Class :character   1st Qu.: 1.75
##  Median :0.04   Mode  :character   Median :14.50
##  Mean   :0.04                      Mean   :20.44
##  3rd Qu.:0.04                      3rd Qu.:34.25
##  Max.   :0.04                      Max.   :63.00
```

Similarly, the function `table()` is a great way to summarise the categorical variables, and highlight any potential mistakes in the dataset. Let's check how many male and female mice there are in our data:

```
table(df$sex)
```

```
##
##  f  F  m  M  N
##  2 37  2 54  1
```

```
# we use the "$" here to specify the column name - how else could we specify the column?
```

Is this output what we expected? Absolutely not! We can see that some entries are lower case, whereas most are upper case. And there is one error - we can assume than "N" was a typo meant to be "M".

Another way we could check the unique values in data is by the aptly named function, `unique()`:

```
unique(df$sex) # unlike table, we don't get an idea of how many observations fall into each category
```

```
## [1] "F" "M" "m" "f" "N"
```

3.2 Navigating and manipulating data frames

Here, we will go through some simple ways you can navigate, and manipulate your data frames. First, let's fix a couple of the issues we identified above:

- Wrong data types for some variables (columns)
- Mistakes in the "sex" variable

Recall that we can convert between some data types. Let's convert one of our "chr" (character) data types into the more appropriate data type "factor". To do this, we need to navigate to the right column. We can do this in several ways:

1. Specifying the column number
2. Specifying the column name
3. Using the "$" symbol and column name

```
# by column number
df[,5]
```

```
##  [1] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Control"
```

```
##  [7] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Treatment"
## [13] "Treatment" "Treatment" "Treatment" "Control"   "Control"   "Control"
## [19] "Control"   "Treatment" "Control"   "Treatment" "Control"   "Control"
## [25] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
## [31] "Control"   "Control"   "Control"   "Treatment" "Control"   "Treatment"
## [37] "Control"   "Treatment" "Control"   "Treatment" "Control"   "Treatment"
## [43] "Control"   "Treatment" "Treatment" "Control"   "Treatment" "Treatment"
## [49] "Treatment" "Control"   "Treatment" "Treatment" "Treatment" "Treatment"
## [55] "Treatment" "Treatment" "Control"   "Control"   "Control"   "Control"
## [61] "Control"   "Control"   "Control"   "Control"   "Treatment" "Treatment"
## [67] "Treatment" "Treatment" "Control"   "Treatment" "Treatment" "Treatment"
## [73] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Treatment"
## [79] "Treatment" "Treatment" "Control"   "Control"   "Treatment" "Control"
## [85] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
## [91] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
```

```
# by column name
df[,"experimental.group"]
```

```
##  [1] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Control"
##  [7] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Treatment"
## [13] "Treatment" "Treatment" "Treatment" "Control"   "Control"   "Control"
## [19] "Control"   "Treatment" "Control"   "Treatment" "Control"   "Control"
## [25] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
## [31] "Control"   "Control"   "Control"   "Treatment" "Control"   "Treatment"
## [37] "Control"   "Treatment" "Control"   "Treatment" "Control"   "Treatment"
## [43] "Control"   "Treatment" "Treatment" "Control"   "Treatment" "Treatment"
## [49] "Treatment" "Control"   "Treatment" "Treatment" "Treatment" "Treatment"
## [55] "Treatment" "Treatment" "Control"   "Control"   "Control"   "Control"
## [61] "Control"   "Control"   "Control"   "Control"   "Treatment" "Treatment"
## [67] "Treatment" "Treatment" "Control"   "Treatment" "Treatment" "Treatment"
## [73] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Treatment"
## [79] "Treatment" "Treatment" "Control"   "Control"   "Treatment" "Control"
## [85] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
## [91] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
```

```
# using $
df$experimental.group
```

```
##  [1] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Control"
##  [7] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Treatment"
## [13] "Treatment" "Treatment" "Treatment" "Control"   "Control"   "Control"
## [19] "Control"   "Treatment" "Control"   "Treatment" "Control"   "Control"
## [25] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
## [31] "Control"   "Control"   "Control"   "Treatment" "Control"   "Treatment"
## [37] "Control"   "Treatment" "Control"   "Treatment" "Control"   "Treatment"
## [43] "Control"   "Treatment" "Treatment" "Control"   "Treatment" "Treatment"
## [49] "Treatment" "Control"   "Treatment" "Treatment" "Treatment" "Treatment"
## [55] "Treatment" "Treatment" "Control"   "Control"   "Control"   "Control"
## [61] "Control"   "Control"   "Control"   "Control"   "Treatment" "Treatment"
## [67] "Treatment" "Treatment" "Control"   "Treatment" "Treatment" "Treatment"
## [73] "Treatment" "Treatment" "Treatment" "Treatment" "Treatment" "Treatment"
## [79] "Treatment" "Treatment" "Control"   "Control"   "Treatment" "Control"
## [85] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
## [91] "Control"   "Control"   "Control"   "Control"   "Control"   "Control"
```

Now, let's convert this to a factor using the `as.factor()` function:

```
# overwrite the "experimental.group" column with itself, just as a different data type
df$experimental.group <- as.factor(df$experimental.group)
```

Check that the conversion has worked:

```
class(df$experimental.group)
```

```
## [1] "factor"
```

Great! Now, to fix the labelling issues with the "sex" column, we need to 1) correct the typo - convert "N" to "M" and 2) convert all values to upper case:

```
# to find the exact row containing the "N" we can use the which() function in
# combination with our square brackets for navigating

df[which(df$sex == "N"),"sex"]
```

```
## [1] "N"
```

```
# now that we can correctly identify the right value
# let's change it to an "M" by assigning it a new value

df[which(df$sex == "N"),"sex"] <- "M"
```

Next, use the `toupper()` function to convert all values within this column to upper case:

```
df$sex <- toupper(df$sex)
```

Let's check that we have fixed both issues!

```
table(df$sex)
```

```
##
##  F  M
## 39 57
```

Perfect!

Next, we will go through a few extra things that might come in handy when manipulating a data frame including:

- Removing a column
- Adding a new column
- Re-ordering a data frame based on a column
- Sub-setting a data frame (splitting it up)
- Joining two data frames together

Removing columns

You may wish to remove certain columns of data, to keep things simple and tidy in future analyses. There are a couple of ways we can do this. The most simple way is to "minus" the column from our existing data frame:

```
new_df <- df[,-5] # create a new data frame, minus column number 5
```

Q: How can we expand this to remove multiple columns??

Another option is to specify the column by name. This takes a little bit more code but is arguably a better way since the code is explicit:

```
new_df <- df[,which(!colnames(df) %in% c("experiment.date","tissue"))]
```

There are a few new things going on here - let's break this down working from the outside in:

- First, we have our df with square brackets meaning we want to navigate somewhere df[]
- The fact that the remaining code is on the right hand side of the comma indicates that we are trying to find a column, not a row [rows,columns]
- The `which()` function helps us navigate to a specific place, so we now have: df[,which()] - i.e. find us the column that satisfies the condition that we specify within the `which()` function
- You can think of the "!" as the word "not", so when we use this in combination with the `colnames()` function, we are saying "not this column"
- The "%in%" symbol means "anything contained within" so when we use !colnames(df) %in% we are saying "NOT any of the column names contained within. . . ."
- To finish the sentence off, we have specified the columns that we want to drop by using c("experiment.date","tissue")

So all together you can read this as. . .

"create a new data frame (new_df) which is the same as the original data frame, but does not contain the columns called"eperimental.date" or "tissue"

df[,which(!colnames(df) %in% c("experiment.date","tissue"))]

Adding columns

Similar to how we can use the "$" to navigate to a particular column, we can use it to create an entirely new one!

Let's say we want to create a new column that combines two of our existing columns: sex and experimental group - this might come in handy if we want to compare experimental groups between males and females:

```
df$new_column <- paste(df$sex, df$experimental.group, sep = "_")

# here we are saying, paste together these two columns, separated by an underscore

# let's look at the new column

head(df)
```

```
##   animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 1        R2   G7      2016.01.12   F          Treatment  liver              10
## 2        R4   G7      2016.01.12   F          Treatment  liver              10
## 3        R1   G9      2016.01.13   M          Treatment  liver              10
## 4        R2   G9      2016.01.13   M          Treatment  liver              10
## 5        R3   G9      2016.01.13   F          Treatment  liver              10
## 6        F4   G9      2016.01.13   M            Control  liver              10
##   homogenate.volume plated.volume         tissue.weight no.cfu  new_column
## 1                 3          0.04   0.14149999999999999     60 F_Treatment
## 2                 3          0.04 6.7900000000000002E-2      0 F_Treatment
## 3                 3          0.04 9.9900000000000003E-2     62 M_Treatment
```

```
## 4                 3          0.04 6.1100000000000002E-2      0 M_Treatment
## 5                 3          0.04 7.4999999999999997E-2     47 F_Treatment
## 6                 3          0.04 2.9399999999999999E-2      0   M_Control
```

Re-ordering a data frame

Sometimes it is useful to re-order the rows of a data frame according to the values in a specific column. We can do this by using the `order()` function:

```
# let's order our df by the column "tissue.weight"

df[order(df$tissue.weight),] [1:10,] # just to print the top 10 rows
```

```
##    animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 20        F3   G9      2016.01.16   F          Treatment  liver             100
## 15        R3   G7      2016.01.12   M          Treatment  liver             100
## 60        R1   G9      2016.01.16   M            Control   lung             100
## 14        F5   G9      2016.01.16   F          Treatment  liver              10
## 11        F1   G9      2016.01.16   F          Treatment  liver              10
## 12        F2   G9      2016.01.16   M          Treatment  liver              10
## 22        R1   G7      2016.01.12   F          Treatment  liver            1000
## 29        R2   G9      2016.01.16   F            Control  liver            1000
## 1         R2   G7      2016.01.12   F          Treatment  liver              10
## 23        F2   G7      2016.01.12   M            Control  liver            1000
##    homogenate.volume plated.volume       tissue.weight no.cfu  new_column
## 20                 3          0.04              0.1045      6 F_Treatment
## 15                 3          0.04              0.1101      3 M_Treatment
## 60                 3          0.04              0.1142     18   M_Control
## 14                 3          0.04 0.12239999999999999     22 F_Treatment
## 11                 3          0.04              0.1245     52 F_Treatment
## 12                 3          0.04 0.12520000000000001     22 M_Treatment
## 22                 3          0.04              0.1376     32 F_Treatment
## 29                 3          0.04              0.1399     39   F_Control
## 1                  3          0.04 0.14149999999999999     60 F_Treatment
## 23                 3          0.04              0.1487      2   M_Control
```

What do you notice?

By default, the `order()` function ranks values in ascending order. If we want the heaviest tissue at the top of the data frame, we can set the argument decreasing=TRUE :

```
df[order(df$tissue.weight, decreasing = TRUE),] [1:10,] # just to print the top 10 rows
```

```
##    animal.id cage experiment.date sex experimental.group tissue dilution.factor
## 61        R2   G9      2016.01.16   F            Control   lung             100
## 3         R1   G9      2016.01.13   M          Treatment  liver              10
## 75        R3   G8      2016.01.16   M          Treatment spleen              10
## 78        F3   G9      2016.01.16   F          Treatment spleen              10
## 16        F1   G9      2016.01.13   M            Control  liver             100
## 77        F2   G9      2016.01.16   M          Treatment spleen              10
## 24        F3   G7      2016.01.12   F            Control  liver            1000
## 30        R4   G9      2016.01.16   F            Control  liver            1000
## 13        F4   G9      2016.01.16   M          Treatment  liver              10
## 53        F2   G9      2016.01.16   M          Treatment   lung              10
##    homogenate.volume plated.volume         tissue.weight no.cfu  new_column
## 61               3.0          0.04                    NA     39   F_Control
## 3                3.0          0.04 9.9900000000000003E-2     62 M_Treatment
```

```
## 75              0.3          0.04 9.7000000000000003E-3      6 M_Treatment
## 78              0.3          0.04 9.7000000000000003E-3      4 F_Treatment
## 16              3.0          0.04               9.69E-2      0   M_Control
## 77              0.3          0.04 9.5999999999999992E-3     27 M_Treatment
## 24              3.0          0.04 9.5000000000000001E-2     13   F_Control
## 30              3.0          0.04               9.35E-2     30   F_Control
## 13              3.0          0.04 9.2999999999999999E-2      5 M_Treatment
## 53              3.0          0.04 9.2999999999999999E-2     11 M_Treatment
```

Sub-setting a data frame

Say we want to perform three separate analyses on this dataset, one of each tissue type (liver, lung, spleen). We can easily create three separate data frames by selecting out the rows of the data frame that we want. This is called sub-setting the data, and we can achieve this by using the `which()` function along with our square brackets for navigation:

```
# let's create a new df containing only the liver data

liver_data <- df[which(df$tissue == "liver"),]

# now one for the spleen data

spleen_data <- df[which(df$tissue == "spleen"),]
```

The "==" is used here to say that "keep rows in the data frame where the values in the tissue column **exactly equal to** to liver".

What if we also only wanted to keep female mice for this experiment? How can we specify both tissue type == liver and sex == F? Intuitively, we can achieve this using the "&" symbol:

```
# subset the original data to keep rows relevant to female liver samples

fem_liver <- df[which(df$tissue == "liver" & df$sex == "F"),]
```

The "&" is part of a group of symbols called logical operators. These are incredibly useful for sub-setting you data and include "==" and "!" which you now know. Others include:

- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to
- | OR
- != not equal to

Q: how could you subset the data to include only spleen samples, with a dilution factor greater than or equal to 100?

As always in R, there are multiple ways to achieve the same thing. There is also an in-built function called `subset()` that you can use to subset your data. Here is an example of how we could use this instead of the `which()` function to subset the data into tissue types:

```
liver_data <- subset(df, df$tissue == "liver")
```

```r
spleen_data <- subset(df, df$tissue == "spleen")

# notice how we don't need to navigate to particular rows for this function to work
# i.e. no square brackets
```

Joining data frames together

You may have imported several data files and want to join them together so that you can perform analysis across the entire data set. Let's assume that our *liver_data* and *spleen_data* represent two different data files, and we want to combine them. A simply way we can do this is using the row bind function `rbind()`:

```r
new_data <- rbind(liver_data, spleen_data)
```

Simple! The caveat here is that both data frames must have identical column names, so that `rbind()` can align the data correctly. If you have data for the same observations (samples) spread across multiple data sets, you can use the function column bind `cbind()` to join them - here, since rows will be matched up the two data frames must have the same row names.

Part 4: Exporting data

In this final part, we will see how to export data back to our working directory. As you may have guessed, the functions for this are very similar to the functions to get data in. Instead of "read" a file, we "write" a file. Let's export our "cleaned" data frame from earlier:

```r
write.csv(df, file = "cleanedData.csv")
```

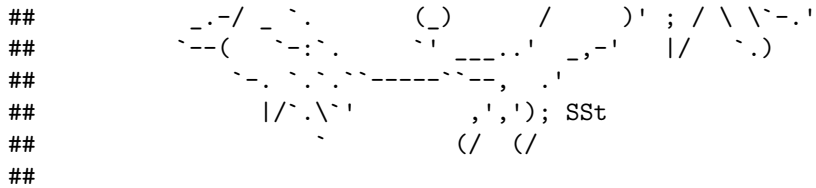
As always, check the help documentation for further details:

```r
?write.csv
```

Similarly, we could use the function `write.table()`

And that's it for today! Congratulations. You can now:
1. Install and load packages
2. Load your data into R
3. Navigate and manipulate your data frames
4. Export data back to your working directory

```
##
##  -----
## See you next time!
##  ------
##     \          ,'``..._   ,'``.
##      \        :,--._:)\,:,._,.:
##       \       :`--,''   :`...';\
##                `,'       `---'  `.
##                /                  :
##               /                    \
##             ,'                      :\.___,-.
##            `...,---'``````-...._    |:       \
##              (                 )   ;:    )   \  _,-.
##               `.              (   //          `'    \
##                :               `.//  )      )     , ;
##              ,-|`.            _,'/       )    ) ,' ,'
##             (  :`.`-..____..=:.-':     .     _,' ,'
##              `,'\ ``--....-)='    `._,  \  ,') _ '``._
```


_.-/ _ `. (_) / )' ; / \ \`-.'
`--( `-:`. `' ___..' _,-' |/ `.)
`-. `.`.``-----``--, .'
|/`.\`' ,','); SSt
` (/ (/