Reproducible Research, Spring 2021

Module 3: Data Basics: Creating, Importing, & Exporting

1 Creating Data

1.1 Vectors

To create a vector of values in R you use the concatenate "c" function.

```
#numeric  vector
nv<-c(2,3,4,2)
print(nv)
```

```
## [1] 2 3 4 2
```

```
#character vector
cv<-c("put","some","words","here")
print(cv)
```

```
## [1] "put"   "some"  "words" "here"
```

```
#logical vector
lv<-c(T,F,F,T)
print(lv)
```

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

1.2 Matrices

The main methods to create matrices are the "matrix" function and the use of the "cbind" or "rbind" functions.

The matrix function takes a vector (creatd with the "c" function) and then provides information on how many rows and/or columns to divide it into.

```
mat<-matrix(c(1,2,3,4,5,6,7,8,9), nrow=3)
print(mat)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

If you supply a number or rows that doesn't evenly divide the elements of the vector and error will appear.

```
matrix(c(1,2,3,4,5,6,7,8,9), nrow=2)
```

```
## Warning in matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 2): data length [9] is
## not a sub-multiple or multiple of the number of rows [2]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8    1
```

Using cbind or rbind to bind columns or rows will also work. If the vectors you are binding are of the same class (numeric, character, logical), the resulting matrix will also be that class. However, if the vectors differ the matrix will default to the most compatible type.

```
mat2<-cbind(nv,cv,lv)
print(mat2)
```

```
##      nv  cv      lv
## [1,] "2" "put"   "TRUE"
## [2,] "3" "some"  "FALSE"
## [3,] "4" "words" "FALSE"
## [4,] "2" "here"  "TRUE"
```

```
mode(mat2)
```

```
## [1] "character"
```

```
mat3<-cbind(nv,lv)
print(mat3)
```

```
##      nv lv
## [1,]  2  1
## [2,]  3  0
## [3,]  4  0
## [4,]  2  1
```

```
mode(mat3)
```

```
## [1] "numeric"
```

1.3 Data Frames

Most imported data will end up a data frame by default. However, you can create a data frame from other R objects using the "data.frame" function. When making a data frame, the columns need to all be the same length and the rows need to be the same length.

```
name<-c("Fred","Bob","Bill","Jim")
weight<-c(129,145,234,198)
height<-c(64,68,72,70)

data<-data.frame(name,weight,height)
print(data)
```

```
##   name weight height
## 1 Fred    129     64
## 2  Bob    145     68
## 3 Bill    234     72
## 4  Jim    198     70
```

1.4 Lists

```
data2<-list(name,weight,height)
print(data2)
```

```
## [[1]]
## [1] "Fred" "Bob"  "Bill" "Jim"
##
## [[2]]
## [1] 129 145 234 198
##
## [[3]]
## [1] 64 68 72 70
```

```
data3<-list(name, weight, height, data)
```

1.5 Converting between types

```
a<-c("10","20","30")
print(a)
```

```
## [1] "10" "20" "30"
```

```
class(a)
```

```
## [1] "character"
```

```
b<-as.numeric(a)
print(b)
```

```
## [1] 10 20 30
```

```
class(b)
```

```
## [1] "numeric"
```

```
c<-as.factor(b)
print(c)
```

```
## [1] 10 20 30
## Levels: 10 20 30
```

```r
class(c)
```

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

2 Importing Data

Getting your data into R can sometimes be the hardest part. Luckily there are packages and functions to help with this.

2.1 Text Files

One of the most common types of data files you might find are deliminated text files (csv, tsv, etc..). These can be read by any text editor, but it requires parsing to turn a text file into data columns. Base R provides read.table and read.csv to do this, but the Tidyverse has better functions in the 'readr' package.

```r
library(readr)

# list of example files, note that csv, tsv, txt, zip, bz2
readr_example()
```

```
##  [1] "challenge.csv"               "chickens.csv"
##  [3] "epa78.txt"                   "example.log"
##  [5] "fwf-sample.txt"              "massey-rating.txt"
##  [7] "mini-gapminder-africa.csv"   "mini-gapminder-americas.csv"
##  [9] "mini-gapminder-asia.csv"     "mini-gapminder-europe.csv"
## [11] "mini-gapminder-oceania.csv"  "mtcars.csv"
## [13] "mtcars.csv.bz2"              "mtcars.csv.zip"
## [15] "whitespace-sample.txt"
```

```r
mtcars_file_location <- readr_example("mtcars.csv")
print(mtcars)
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
```

```
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
# mtcars <- read_csv("/home/bsteves/R/x86_64-pc-linux-gnu-library/4.2/readr/extdata/mtcars.csv")
```

You can enter the file location manually for example..

```
mtcars <- read_csv("/home/bsteves/R/x86_64-pc-linux-gnu-library/4.2/readr/extdata/mtcars.csv")
```

```
## Rows: 32 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (11): mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

But this will vary from system to system and will need to be edited to work on your system. You can also just specify file location by wrapping read_csv around readr_example()

```
mtcars <- read_csv(readr_example("mtcars.csv"))
```

```
## Rows: 32 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (11): mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Readr guesses based on the first 1000 records.
What if the first 1000 records are NA for a column? Here is an example of data that doesn't get guessed properly.

```
df1 <- read_csv(readr_example("challenge.csv"))
```

```
## Rows: 2000 Columns: 2
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## dbl  (1): x
## date (1): y
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# check out problems
problems(df1)
```

```
## # A tibble: 0 x 5
## # i 5 variables: row <int>, col <int>, expected <chr>, actual <chr>, file <chr>
```

```
# change guess to use 1001 rows
df2 <- read_csv(readr_example("challenge.csv"), guess_max = 1001)
```

```
## Rows: 2000 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (1): x
## date (1): y
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Or specifiy the columns
df3 <- read_csv(
  readr_example("challenge.csv"),
  col_types = cols(
    x = col_double(),
    y = col_date(format = "")
  )
)
```

readr has many options for parsing out column names, setting column definitions, and even skipping rows when needed (e.g. sometimes metadata exists at the top of text data files)

Note that csv files in a zip can be read as well.

```
mtcars2 <- read_csv(readr_example("mtcars.csv.zip"))
```

```
## Rows: 32 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (11): mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dat <- "a,b,c
1,2,3
4,5,6"

print(dat)
```

```
## [1] "a,b,c\n1,2,3\n4,5,6"
```

```
read_csv(dat)
```

```
## Rows: 2 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): a, b, c
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 2 x 3
##       a     b     c
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

```
dat <- read_csv("a,b,c
1,2,3
4,5,6")
```

```
## Rows: 2 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): a, b, c
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Quick comparison in speed between base R's read.csv() and readr's read_csv(). The usage of read.csv() is very similar to read_csv().

```
customer_dat<-read.csv("data/customers-100000.csv")

# read.csv time
system.time(read.csv("data/customers-100000.csv"))
```

```
##    user  system elapsed
##   0.824   0.004   0.827
```

```
# read_csv time
system.time(read_csv("data/customers-100000.csv"))
```

```
## Rows: 100000 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (10): Customer Id, First Name, Last Name, Company, City, Country, Phone...
## dbl   (1): Index
## date  (1): Subscription Date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
##    user  system elapsed
##   0.507   0.008   0.410
```

Beyond the speed increase, here are a couple other benefits to using readr over base R for text file importing.

1.) readr returns a tibble, which is a tidyverse improvement over a plain data.frame. Tibbles don't mess up your column names, don't force characters into factors, and display better in console output.

2.) readr output is more reproducible. Base R's read.csv() inherits some behavior from your operating system, but readr does not.

2.2 Copy/Paste

If you try to copy and paste some data in from a text file or an Excel spreadsheet and assign it to an object you just end up with a character vector with a single element made up of long string of characters.

```
excel_copy_paste_dat <- "A  B   C
1   2   3
7   6   5
"

print(excel_copy_paste_dat)
```

```
## [1] "A \tB \tC\n1\t2\t3\n7\t6\t5\n"
```

Probably not what you wanted.

It looks similar to the CSV file we just hand wrote and read in using read_csv. Let's try that.

```
read_csv(excel_copy_paste_dat)
```

```
## Rows: 2 Columns: 1
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): A   B   C
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 2 x 1
##   `A \tB \tC`
##   <chr>
## 1 "1\t2\t3"
## 2 "7\t6\t5"
```

Nope, that's not it. Looks like it's deliminated by '^' and not commas. Those are tabs, and readr has read_tsv().

```
read_tsv(excel_copy_paste_dat)
```

```
## Rows: 2 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## dbl (3): A, B, C
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 2 x 3
##       A     B     C
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     7     6     5
```

Much better. Use "read_tsv()" when copying and pasting Excel data. It's important to figure out what the deliminator is when using these functions. In fact, read_delim() is a function that lets you set the deliminator. Also, if you come across European style csv files (';' deliminated with ',' for decimals) you can use read_csv2().

2.3 Excel Files

The readxl package handles opening Excel files (xls, xlsx).
Like readr, the package comes with an example workbook and the readxl_example() function returns the location of that file on your system.

```
library(readxl)
datasets <- readxl_example("datasets.xlsx")
```

This datasets.xlsx is an Excel file, with a different data set on each sheet. Use excel_sheets() function to return list of sheets

```
excel_sheets(datasets)
```

```
## [1] "iris"     "mtcars"   "chickwts" "quakes"
```

Use the read_excel() function to return a sheet, either by name or sheet number

```
iris <- read_excel(datasets, 1)
head(iris)
```

```
## # A tibble: 6 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <chr>
## 1          5.1         3.5          1.4         0.2 setosa
## 2          4.9         3            1.4         0.2 setosa
## 3          4.7         3.2          1.3         0.2 setosa
## 4          4.6         3.1          1.5         0.2 setosa
## 5          5           3.6          1.4         0.2 setosa
## 6          5.4         3.9          1.7         0.4 setosa
```

```
chickwts <- read_excel(datasets, "chickwts")
head(chickwts)
```

```
## # A tibble: 6 x 2
##   weight feed
##    <dbl> <chr>
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

You can even specify a cell range. Don't forget that column labels are in cells. Here we'll grab columsn A through D and the first 11 rows (column labels and 10 rows of data)

```
sel_iris <- read_excel(datasets, "iris", range="A1:D11")
sel_iris
```

```
## # A tibble: 10 x 4
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
##           <dbl>       <dbl>        <dbl>       <dbl>
##  1          5.1         3.5          1.4         0.2
##  2          4.9         3            1.4         0.2
##  3          4.7         3.2          1.3         0.2
##  4          4.6         3.1          1.5         0.2
##  5          5           3.6          1.4         0.2
##  6          5.4         3.9          1.7         0.4
##  7          4.6         3.4          1.4         0.3
##  8          5           3.4          1.5         0.2
##  9          4.4         2.9          1.4         0.2
## 10          4.9         3.1          1.5         0.1
```

2.4 Database Connection

Connections can be made directly with a database. However, which databases you can connect to is dependent on your operating system. For example, you need to be using Windows to connect to a MS Access database.

```
library(RMySQL)
```

```
## Loading required package: DBI
```

```
# create a RMySQL.cnf file.  This is set up ahead of time on your computer.

con <- RMySQL::dbConnect(MySQL(), group = 'GlobalInvHist_be_local')


# You could also connect directly with the connection string.
# This is just an example and will throw an error on my computer because those credentials are fake.
```

```
con <- RMySQL::dbConnect(MySQL(),
                         user = "myusername",
                         password = "mypassword",
                         dbname = "GlobalInvHist_be",
                         host = "localhost")
```

```
## Error in .local(drv, ...): Failed to connect to database: Error: Access denied for user 'myusername'@
```

This isn't very secure to save passwords in a script, esp. if you share your scripts on places like GitHub. This example will throw an error on my computer because those credentials are fake.

In Rstudio, you can set up prompts to ask for username and/or password

```
con <- RMySQL::dbConnect(MySQL(),
                         user = rstudioapi::askForPassword("Database user"),
                         password = rstudioapi::askForPassword("Database password"),
                         dbname = "GlobalInvHist_be",
                         host = "localhost")
```

In newer versions of RStudio, there is a 'connections' tab in the upper right hand panel. This will let you explore databases, tables, columns of your database connection. Alternatively, you can access that information with in the console.

```
# list tables.
dbListTables(con)
```

```
##  [1] "Alerts"                    "BroadRanges"
##  [3] "Citations"                 "Codes"
##  [5] "CommonNames"               "Dist2"
##  [7] "Distributions"             "EPV"
##  [9] "Ecology"                   "EcologyKeywords"
## [11] "EcologyParameterValues"    "EcologyParameterValues_old"
## [13] "Glossary"                  "Groups"
## [15] "Habitats"                  "HigherTaxonomy"
## [17] "Images"                    "Impacts"
## [19] "Journals_old"              "NativeRanges"
## [21] "NewsItems"                 "NumInvPerReg"
## [23] "Occ2"                      "Occurrences"
## [25] "Occurrences2"              "Occurrences_bckup"
## [27] "PICES_LifeHist"            "PICES_Occ"
## [29] "Polygons"                  "Potential_MisIDs"
## [31] "References_old"            "Regions"
## [33] "SpeciesComments"           "SpeciesComments2"
## [35] "Synonyms"                  "Table1"
## [37] "Table2"                    "Taxonomy"
## [39] "Vectors"                   "WORMS_Common_Names"
## [41] "WORMS_Synonyms"            "WORMS_Taxonomy"
## [43] "WebLinks"                  "commeco"
## [45] "countries"                 "dailyinvaders"
## [47] "economy"                   "invhst"
## [49] "journals"                  "lfhsp"
## [51] "new_occ"                   "old_occ"
## [53] "refs"                      "spimgs"
```

```
## [55] "splinks"                    "tables_fields"
## [57] "taxa"                       "taxa_cits"
## [59] "taxanew"                    "taxon"
## [61] "tbl_factsheets"             "tbl_species_new"
## [63] "users"
```

```
# list fields in a table
dbListFields(con, "Taxonomy")
```

```
##  [1] "TXA_ITIS"           "TXA_Level"          "TXA_Name"
##  [4] "TXA_Binomial"       "TXA_Parent"         "TXA_Author"
##  [7] "TXA_Year"           "TXA_CAAB"           "USER"
## [10] "TXA_Time"           "TXA_Group"          "TXA_ReleaseDate"
## [13] "TXA_ExpReleaseDate" "TXA_Created_At"     "TXA_Updated_At"
## [16] "TXA_WoRMsID"        "TXA_ProfileStatus"  "created"
## [19] "created_by"         "updated"            "updated_by"
```

```
# return a whole table
taxanew <- dbReadTable(con, "Taxonomy")
```

```
## Warning in dbSendQuery(conn, statement, ...): unrecognized MySQL field type 7
## in column 17 imported as character
```

```
## Warning in dbSendQuery(conn, statement, ...): unrecognized MySQL field type 7
## in column 19 imported as character
```

```
# get results of a query to a table
sel_taxa <- dbGetQuery(con, "SELECT count(*) FROM Taxonomy WHERE TXA_Binomial IS NOT NULL;")
```

You can also send a query using dbSendQuery() for advanced queries like delete, modify, etc.. Sometimes you don't want to return all of the records at once. In those cases you can use dbSendQuery() and then fetch() the results as you need them. I generally don't use the dbSendQuery() and fetch() method.

2.5 Other formats (JSON, XML, etc..)

Sometimes you'll find data in other formats. It's worth looking to see if a package has been made to help import that kind of data.

3 Exporting Data

3.1 Text Files

An analog to read.csv is write.csv. You can write just about any data frame into a csv file with this command

```
write_csv(taxanew, "data/taxanew.csv")
```

3.2 Excel Files

The readxl package doesn't have a write_excel() function. But you can always just use write_csv() and save to a csv file that Excel can open.

3.3 Database Connection

If you have the proper permissions, you can run append, insert, and update queries using a connection to the database. You can also write a data frame into a table using "dbWriteTable(con,"table name", a.data.frame).

Writing data to a database table. Using the existing connection and dataframe from before.

```
dbSendQuery(con, "SET GLOBAL local_infile = true;") # this line gives us permission to load data from a
```

```
## <MySQLResult:0,1,4>
```

```
dbWriteTable(con, "taxanew", taxanew, overwrite=TRUE)
```

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

Options for dbWriteTable include "append=TRUE", "overwrite=TRUE", "row.names=FALSE"

3.4 Other formats (JSON, XML, etc..)

The same specialized R packages you used to import other types of data will likely have a function for writing that type of data from R objects.

3.5 dput function

It is often useful to useful to export an R object into an R statement that can be used to recreate that object elsewhere. For example, say you wanted to ask an R question on http://stackoverflow.com/ and needed to include some small snippet of your data to help explain your issue. Just doing a copying and pasting will cause formatting issues and it will be difficult for others to use your data to help you out. However, if we use the "dput" function we can transform an R object into a the command we'd need to recreate that structure.

```
first_5_taxanew <- head(taxanew, 5)
dput(first_5_taxanew)
```

```
## structure(list(TXA_ITIS = -1153:-1149, TXA_Level = c(143L, 143L,
## 140L, 143L, 323L), TXA_Name = c("panamensis", "philippina", "Aglaeopheniidae",
## "turdus", "Naria"), TXA_Binomial = c("Nereis panamensis", "Macrorhynchia philippina",
## NA, "Naria turdus", NA), TXA_Parent = c(165902L, 50409L, 718926L,
## -1149L, 72741L), TXA_Author = c("Fauchald 1877", "Kirchenpauer, 1872 AphiaID",
## "Marktanner-Turneretscher,", "Lamarck 1810", "Gray 1837"), TXA_Year = c(NA_character_,
## NA_character_, NA_character_, NA_character_, NA_character_),
##     TXA_CAAB = c(NA_integer_, NA_integer_, NA_integer_, NA_integer_,
##     NA_integer_), USER = c(NA_character_, NA_character_, NA_character_,
##     NA_character_, NA_character_), TXA_Time = c(NA_character_,
##     NA_character_, NA_character_, NA_character_, NA_character_
##     ), TXA_Group = c("Arthropoda", "Cnidarians-Hydrozoans", "Cnidarians-Hydrozoans",
##     "Mollusks-Gastropods", "Mollusks-Gastropods"), TXA_ReleaseDate = c("2099-01-01",
##     "2099-01-01", "2099-01-01", "2099-01-01", "2099-01-01"),
##     TXA_ExpReleaseDate = c(NA_character_, NA_character_, NA_character_,
##     NA_character_, NA_character_), TXA_Created_At = c(NA_character_,
##     NA_character_, NA_character_, NA_character_, NA_character_
##     ), TXA_Updated_At = c(NA_character_, NA_character_, NA_character_,
```

```
##     NA_character_, NA_character_), TXA_WoRMsID = c(NA_integer_,
##     NA_integer_, NA_integer_, NA_integer_, NA_integer_), TXA_ProfileStatus = c(NA_character_,
##     NA_character_, NA_character_, NA_character_, NA_character_
##     ), created = c("2024-02-02 10:39:30", "2024-02-02 10:39:30",
##     "2024-02-02 10:39:30", "2024-02-02 10:39:30", "2024-02-02 10:39:30"
##     ), created_by = c(NA_integer_, NA_integer_, NA_integer_,
##     NA_integer_, NA_integer_), updated = c("2024-02-02 10:39:31",
##     "2024-02-02 10:39:31", "2024-02-02 10:39:31", "2024-02-02 10:39:31",
##     "2024-02-02 10:39:31"), updated_by = c(NA_integer_, NA_integer_,
##     NA_integer_, NA_integer_, NA_integer_)), row.names = c(NA,
## 5L), class = "data.frame")
```

If I copy the output to that and assign it to an object..

```
taxanew<-structure(list(id = c(-578L, -577L, -576L, -575L, -574L, -572L,
-571L, -570L, -569L, -568L), binomial = c("Dasya sp. A", "Dasya sessilis",
"Pkea yoshizakii", "Chondracanthus teedei", NA, "Tricellaria inopinata",
"Pachycordyle michaeli", "Gambusia holbrooki", "Corella inflata",
NA), taxa_group = c("Crustaceans-Copepods", "Algae", "Algae",
"Algae", NA, "Ectoprocts", "Coelenterates-Hydrozoans", "Fishes",
"Tunicates", NA)), .Names = c("id", "binomial", "taxa_group"), row.names = c(NA,
10L), class = "data.frame")


# I get a copy of the original object
print(taxanew)
```

```
##      id              binomial               taxa_group
## 1  -578           Dasya sp. A     Crustaceans-Copepods
## 2  -577        Dasya sessilis                   Algae
## 3  -576       Pkea yoshizakii                   Algae
## 4  -575 Chondracanthus teedei                   Algae
## 5  -574                  <NA>                    <NA>
## 6  -572 Tricellaria inopinata               Ectoprocts
## 7  -571 Pachycordyle michaeli Coelenterates-Hydrozoans
## 8  -570    Gambusia holbrooki                  Fishes
## 9  -569       Corella inflata                Tunicates
## 10 -568                  <NA>                    <NA>
```

Homework

1. Import the data set you found for your last homework into R.
2. Submit both your data set (or small part of it) and an R script of the code you used to import it to my in Teams.