Efficient Management of Data in R (Data Structures!)

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

11/15/2023

Importing data

The first problem a data scientist will usually face is how to import data into R!

Often they have to import data from either a file, a database, or other sources. One of the most common ways of storing and sharing data for analysis is through electronic spreadsheets.

A spreadsheet stores data in rows and columns. It is basically a file version of a `data frame` (or a `tibble!`).

Importing data

A common function for importing data is the `read.table` function:

```
mydata <- read.table("mydata.txt")
```

This is looking for a structured dataset, with the same number of entries in each row, and data that is delimited with a single space between values.

Importing data

The `read.table` function can also read tab-delimited data:

```
mydata <- read.table("mydata.txt", sep="\t")
```

Or comma separated (.csv) formats:

```
mydata <- read.table("mydata.txt", sep=",")
```

(also explore the `read.csv` function)

Importing data

We can also add options to set the first column as a header and select a row for the row labels:

```
mydata <- read.table("mydata.txt",
                     header=TRUE,
                     row.names="id")
```

Importing data

Excel files can also be directly imported using `read.xlsx`:

```
library(xlsx)
mydata <- read.xlsx("myexcel.xlsx")
```

And one can also select a specific sheet in the Excel file:

```
mydata <- read.xlsx("myexcel.xlsx",
                    sheetName = "mysheet")
```

Other functions for importing data

Other useful importing tools are `scan`, `readLines`, `readr`, and `readxl`. The latter two we will discuss later.

Exporting data

We have many options for exporting data from R. For data frames, one of the easiest ways to output data is with the `write.table` function:

```
write.table(dat, file = "data_out.txt",
            quote = FALSE, sep = ",",
            row.names = TRUE,
            col.names = TRUE)
```

Exporting data

Another important and useful way of inputting/outputting data is in an Rds object:

```
saveRDS(dat, file = "dat.Rds")
dat.copy <- readRDS(file = "dat.Rds")
```

Importance of data structures

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks.

Data structures in R programming are tools for holding multiple values, variables, and sometimes functions.

Please think very carefully about the way you manage and store your data! This can make your life much easier and make your code and data cleaner and more portable!

Types of data structures in R

R's base data structures are often organized by their dimensionality (1D, 2D, nD) and whether they're homogeneous or heterogeneous (elements of identical or various type). Six of the most common data types are:

1. Vectors
2. Lists
3. Matrices
4. Arrays
5. Factors
6. Data frames (or tibbles)

Data Frames

The most common data structure for storing a dataset in R is in a **data frame**. Conceptually, we can think of a data frame as a two dimensional table with rows representing observations and the different variables reported for each observation defining the columns. Data frames are particularly useful for datasets because we can combine different data types into one object.

Data Frames

We can convert matrices into data frames using the function `as.data.frame`:

```
mat <- matrix(1:12, 4, 3)
mat <- as.data.frame(mat)
```

Or just generate it directly using the `data.frame` function:

```
dat <- data.frame(x=1:4, y=5:8, z=9:12)
```

A `data.frame` can be indexed as matrices, `dat[1:2, 2:3]`, and columns can be extracted using the `$` operator.

Tibbles

Here is a printed version of the data frame:

```
dat
```

```
##   x y  z
## 1 1 5  9
## 2 2 6 10
## 3 3 7 11
## 4 4 8 12
```

Tibbles

A **tibble** is a modern version of a data.frame.

```
library(tidyverse)
dat1 <- tibble(x=1:4, y=5:8, z=9:12)
```

Or convert a data.frame to a tibble

```
dat <- data.frame(x=1:4, y=5:8, z=9:12)
dat1 <- as_tibble(dat)
```

Tibbles

Here is a printed version of the tibble:

```
dat1
```

```
## # A tibble: 4 x 3
##       x     y     z
##   <int> <int> <int>
## 1     1     5     9
## 2     2     6    10
## 3     3     7    11
## 4     4     8    12
```

Tibbles

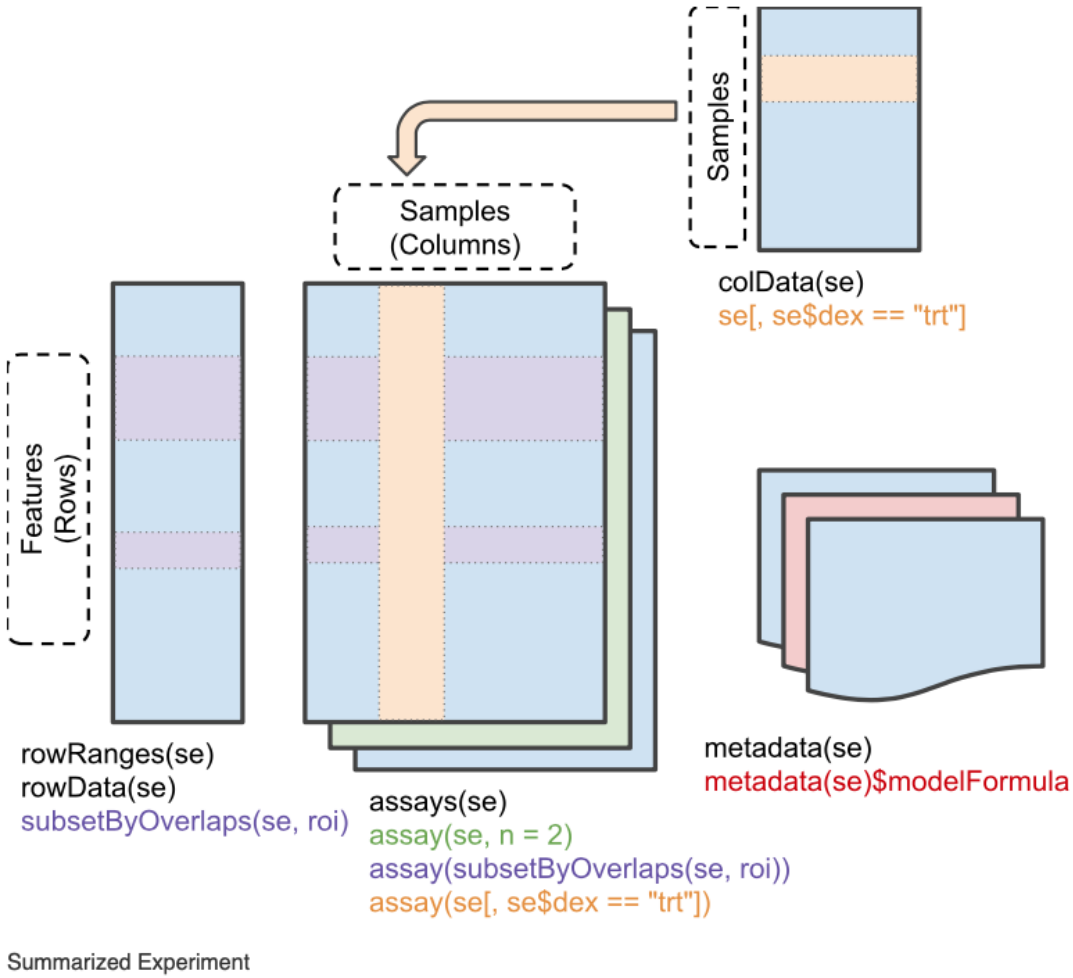
Important characteristics that make tibbles unique:

1. Tibbles are primary data structure for the `tidyverse`
2. Tibbles display better and printing is more readable
3. Tibbles can be grouped
4. Subsets of tibbles are tibbles
5. Tibbles can have complex entries–numbers, strings, logicals, lists, functions.
6. Tibbles can (almost) enable object-orientated programming in R

Advanced Data Structures in R

Aso you gain more experience in R, you should explore more advanced R data structures, namely the **S3** and **S4** class objects. These can facilitate object orientated programming.

One example of an S4 class data structure is the **SummarizedExperiment** object.



Session info

```
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.5.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK vers
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] lubridate_1.9.3 forcats_1.0.0   stringr_1.5.0   dplyr_1.1.3
##  [5] purrr_1.0.2     readr_2.1.4     tidyr_1.3.0     tibble_3.2.1
##  [9] ggplot2_3.4.4   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4      compiler_4.3.2    tidyselect_1.2.0  scales_1.2.1
##  [5] yaml_2.3.7        fastmap_1.1.1     R6_2.5.1          generics_0.1.3
##  [9] knitr_1.45        munsell_0.5.0     pillar_1.9.0      tzdb_0.4.0
## [13] rlang_1.1.2       utf8_1.2.4        stringi_1.8.1     xfun_0.41
## [17] timechange_0.2.0  cli_3.6.1         withr_2.5.2       magrittr_2.0.3
## [21] digest_0.6.33     grid_4.3.2        rstudioapi_0.15.0 hms_1.1.3
## [25] lifecycle_1.0.4   vctrs_0.6.4       evaluate_0.23     glue_1.6.2
## [29] fansi_1.0.5       colorspace_2.1-0  rmarkdown_2.25    tools_4.3.2
## [33] pkgconfig_2.0.3   htmltools_0.5.7
```