ISA 401: Business Intelligence & Data Visualization

03: Importing and Exporting Data in R

Fadel M. Megahed, PhD

Professor of Information Systems and Business Analytics
Farmer School of Business
Miami University

@FadelMegahed
fmegahed
fmegahed@miamioh.edu
Automated Scheduler for Office Hours

Fall 2024

Quick Refresher from Last Class

- ✅ Describe why we are using R in this course?
- ✅ Understand the syntax, data structures and functions
- ✅ Utilize the project workflow in R and create your second R script.

Going Over Assignment 02 Solutions

Let us go over the solutions for assignment 02 together.

Learning Objectives for Today's Class

- Subset data in R.
- Read text-files, binary files (e.g., Excel, SAS, SPSS, Stata, etc), json files, etc.
- Export data from R.

Subsetting Data

Recall: Atomic Vectors (1D)

Atomic vectors are 1D data structures in R, where all elements **must have the same type.**

Since they are **1D data structures**, they are subsetted using `[element_no(s)]`.

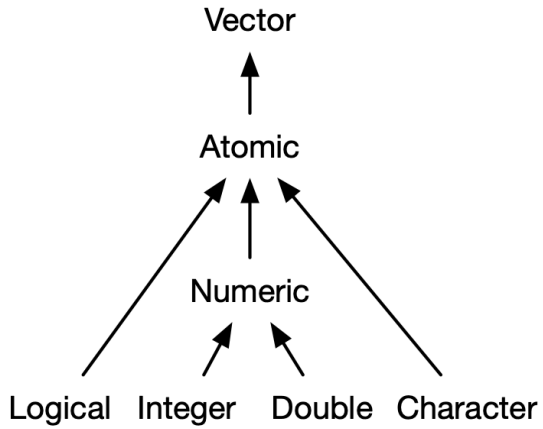
```
x_vec = rnorm(3)
x_vec
```

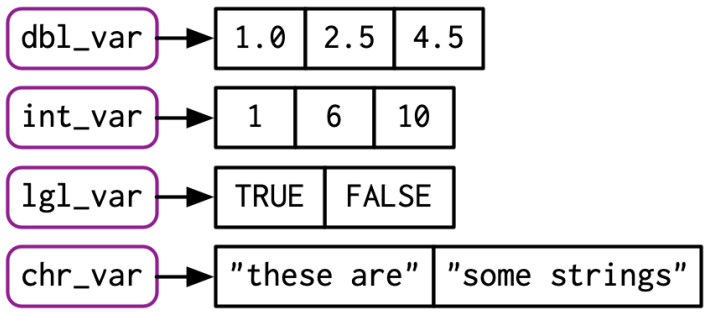
```
## [1] -0.9393151  0.1021115  0.4393175
```

```
x_vec[2]
```

```
## [1] 0.1021115
```

```
x_vec[c(1,3)]
```





Sources: Images are from Hadley Wickham's Advanced R Book, Chapter 3 on Vectors.

Recall: Lists

```
lst <- list( 1:5, "a", c(TRUE, FALSE, TRUE), c(2.3, 5.9) )
```

Subset by []

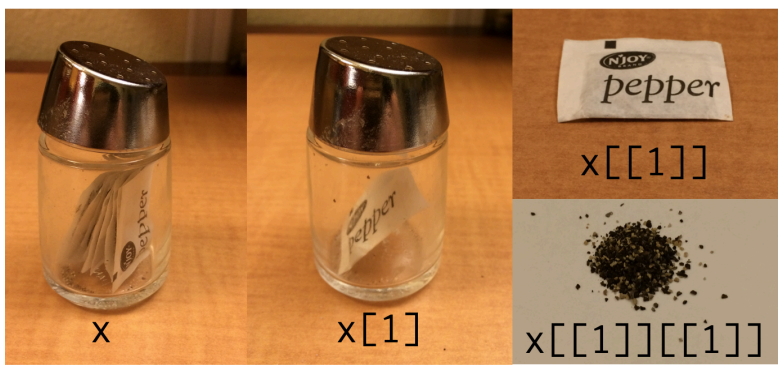
```
lst[4]
```

```
## [[1]]
## [1] 2.3 5.9
```

Subset by [[]]

```
lst[[4]]
```

```
## [1] 2.3 5.9
```



Sources: Image is from Hadley Wickham's Tweet on Indexing lists in R.

Recall: Matrices (2D)

A matrix is a **2D data structure** made of **one/homogeneous data type.**

A 2 × 2 numeric matrix

```
x_mat = matrix( sample(1:10, size = 4), n
```

```
x_mat # printing it nicely
print('-----------------')
x_mat[1, 2] # subsetting
```

```
##      [,1] [,2]
## [1,]   10    6
## [2,]    1    2
## [1] "-----------------"
## [1] 6
```

A 3 × 4 character matrix

```
x_char = matrix( sample(letters, size = 1
x_char
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "r"  "k"  "j"  "t"
## [2,] "b"  "o"  "v"  "p"
## [3,] "f"  "c"  "l"  "w"
```

```
x_char[1:2, 2:3] # subsetting
```

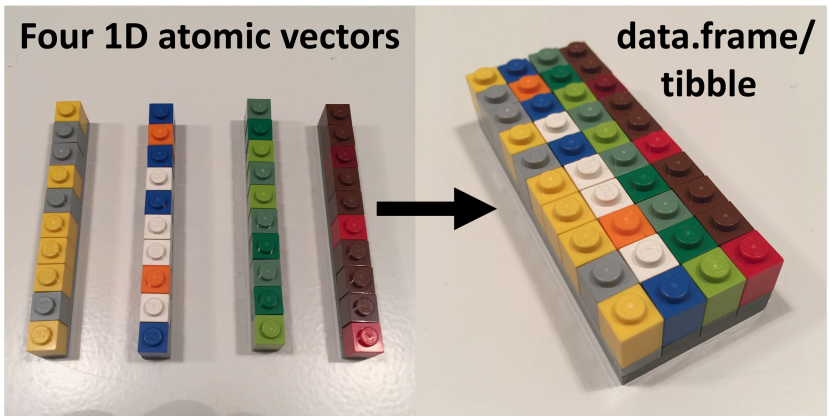
```
##      [,1] [,2]
## [1,] "k"  "j"
## [2,] "o"  "v"
```

Tibbles

```
dept = c('ACC', 'ECO', 'FIN', 'ISA', 'MGM
nfaculty = c(18L, 19L, 14L, 25L, 22L)

fsb_tbl <- tibble::tibble(
  department = dept,
  count = nfaculty,
  percentage = count / sum(count))
fsb_tbl
```

```
## # A tibble: 5 × 3
##   department count percentage
##   <chr>      <int>      <dbl>
## 1 ACC           18      0.184
## 2 ECO           19      0.194
## 3 FIN           14      0.143
## 4 ISA           25      0.255
## 5 MGMT          22      0.224
```



Source: The image is from the excellent lego-rstats GitHub Repository by Jenny Bryan

Subsetting Tibbles

to 1d

- with `[[]]` or `$`

```
fsb_tbl[["count"]] # column name
```

```
## [1] 18 19 14 25 22
```

```
fsb_tbl[[2]] # column position
```

```
## [1] 18 19 14 25 22
```

```
fsb_tbl$count # column name
```

```
## [1] 18 19 14 25 22
```

Source: Slide is based on Earo Wang's STAT 220 Slides.

Subsetting Tibbles

by columns

- with `[]` or `[, col]`

```
fsb_tbl["count"]
```

```
## # A tibble: 5 × 1
##   count
##   <int>
## 1    18
## 2    19
## 3    14
## 4    25
## 5    22
```

```
fsb_tbl[2] # for data.frames -
```

```
## # A tibble: 5 × 1
##   count
##   <int>
## 1    18
## 2    19
## 3    14
## 4    25
## 5    22
```

Subsetting Tibbles

by rows

- with `[row, ]`

```
fsb_tbl[c(1, 3), ]
```

```
## # A tibble: 2 × 3
##   department count percentag
##   <chr>      <int>      <db
## 1 ACC           18      0.18
## 2 FIN           14      0.14
```

```
fsb_tbl[-c(2, 4), ]
```

```
## # A tibble: 3 × 3
##   department count percentag
##   <chr>      <int>      <db
## 1 ACC           18      0.18
## 2 FIN           14      0.14
## 3 MGMT          22      0.22
```

Subsetting Tibbles

by rows and columns

- with `[row, col]`

```
fsb_tbl[1:3, 2:3]
## ## fsb_tbl[-4, 2:3] # same as above
## ## fsb_tbl[1:3, c("count". "percentage")] # same result
## ## fsb_tbl[c(rep(TRUE, 3), FALSE), 2:3] # same as above
```

```
## # A tibble: 3 × 2
##   count percentage
##   <int>      <dbl>
## 1    18      0.184
## 2    19      0.194
## 3    14      0.143
```

Subsetting Tibbles

- Use `[[` to extract 1d vectors from 2d tibbles
- Use `[` to subset tibbles to a new tibble
  - numbers (positive/negative) as indices
  - characters (column names) as indices
  - logicals as indices

```
fsb_tbl[["count"]] # will produce 1-D vector
fsb_tbl$count # will produce 1D vector

# Resulting in tibbles
fsb_tbl[, 2]
fsb_tbl[1:3, 2:3]
```

Source: Slide is based on Earo Wang's STAT 220 Slides.

Data import ⬇️

Reading Plain-Text Rectangular

(a.k.a. flat or spreadsheet-like files)

- delimited text files with `read_delim()`
  - `.csv`: comma separated values with `read_csv()`
  - `.tsv`: tab separated values `read_tsv()`
- `.fwf`: fixed width files with `read_fwf()`

Some Details on Reading CSV Data Files

`read_csv()` arguments with `?read_csv()`


```
readr::read_csv(
  file,
  col_names = TRUE,
  col_types = NULL,
  locale = default_locale(),
  na = c("", "NA"),
  quoted_na = TRUE,
  quote = "\"",
  comment = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  progress = show_progress(),
  skip_empty_rows = TRUE
)
```

Demo: Reading CSV Data

In this hands-on demo, you will learn how to:

- Import CSV files into your R environment based on:
  - files that are located on your computer, see **Canvas** for downloading an example CSV
  - files that are hosted on the web.
    - **Data in Webpages:** we will cover the following example in class:
      - **FRED Data:** e.g., Unempolyment Rate (UNRATE)
    - **GitHub** Repositories, e.g.,
      - SuperBowl Ads
      - Women's Rights Around the World - focusing on `WomenTotal.csv`

Advanced: Reading CSVs with the vroom

Faster delimited reader at **1.4GB/sec**

- vroom is a relatively new `tidyverse` package that can **read** and **write** delimited files very efficiently.
- It is recommended for large CSV files, see tidyverse blog for a detailed introduction on the package.

```
if(require(vroom)==FALSE) install.packages('vroom')
fast_df <- vroom::vroom("your_file.csv")
```

03:00

Reading Proprietary Binary Files

Microsoft Excel (with extensions `.xls`for MSFT Excel 2003 and earlier **OR** `.xlsx` for MSFT Excel 2007 and later)

Non-Graded Class Activity

Activity | Your Solution | My Solution

- Download the AIAAIC Repository.xlsx file from Canvas.
- Store the data in an appropriate location on your computer (e.g., within the data folder for ISA 401)
- Use an appropriate function from the `readxl` package to read the data (either `read_xlsx()` or `read_xls()`).
- Report the number of observations, variables and the class of each variable from the data.

03:00

Reading Proprietary Binary Files

Microsoft Excel (with extensions `.xls`for MSFT Excel 2003 and earlier **OR** `.xlsx` for MSFT Excel 2007 and later)

Non-Graded Class Activity

Activity | Your Solution | My Solution

> *Over the next 3 minutes, use an R script file to answer the questions from the activity and record your answers below*

Number of observations and variables: ..... and .....

The class of each variable .....

03:00

Reading Proprietary Binary Files

Microsoft Excel (with extensions `.xls`for MSFT Excel 2003 and earlier **OR** `.xlsx` for MSFT Excel 2007 and later)

Non-Graded Class Activity

Activity | Your Solution | My Solution

Please refer to our discussion in class

Reading Proprietary Binary Files

Several functions from the haven 📦 can be used to read and write formats used by other statistical packages. Example functions include:

- SAS
  - `.sas7bdat` with `read_sas()`
- Stata
  - `.dta` with `read_dta()`
- SPSS
  - `.sav` with `read_sav()`

Please refer to the help files for each of those packages for more details.

JSON Files

> *JSON (JavaScript Object Notation) is an open standard file format and data interchange format that uses **human-readable** text to store and transmit data **objects** consisting of **attribute–value pairs** and **arrays**... It is a common data format with diverse uses ... including that of web applications with servers.* --- Wikipedia's Definition of JSON

- **object:** `{}`
- **array:** `[]`
- **value:** string/character, number, object, array, logical, `null`

JSON Files

JSON

```
{
  "firstName": "Mickey",
  "lastName": "Mouse",
  "address": {
    "city": "Mousetown",
    "postalCode": 10000
  }
  "logical": [true, false]
}
```

R list

```
list(
  firstName = "Mickey",
  lastName = "Mouse",
  address = list(
    city = "Mousetown",
    postalCode = 10000
  ),
  logical = c(TRUE, FALSE)
)
```

Demo

We will use the jsonlite 📦 to read an example from one of the awesome-json-datasets.

Please note the following from the demo:

- **Setting up the package**, which should be a one-time event if you are using the same computer.
- **Which function** are we using from the package to read the json data?
- What is the **type of object returned** by the function?
- How are we **converting the object to a tibble?**

Data export ⬆️

From Read to Write

`read_*()` to `write_*()`

Here are some ideas: **do they come from the same package?**

```
readr::write_csv(example_tbl, file = "example.csv")
haven::write_sas(example_tbl, path = "example.sas7bdat")
jsonlite::write_json(example_tbl, path = "example.json")
```

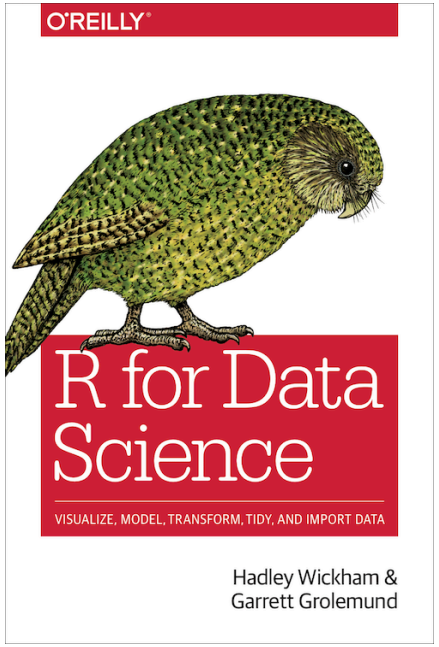
Recap

Summary of Main Points

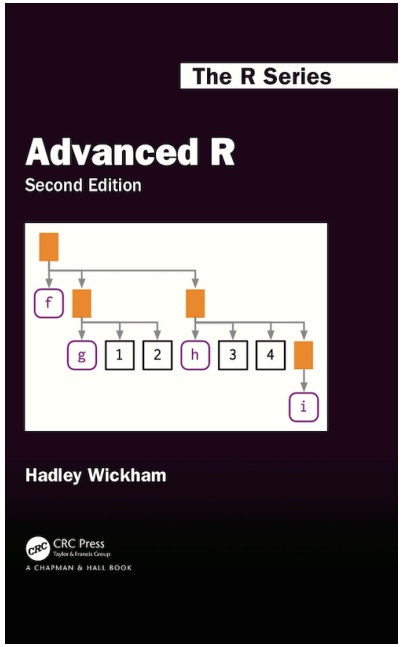
By now, you should be able to do the following:

- Subset data in R.
- Read text-files, binary files (e.g., Excel, SAS, SPSS, Stata, etc), json files, etc.
- Export data from R.

Supplementary Reading



- Tibbles
- Data import



- Subsetting

Things to Do to Prepare for Our Next Class

- Go over your notes and complete Assignment 03 on Canvas.
- **Before attempting the assignment, you are encouraged to:**
  - Go over this slide deck as well as the slide deck from last class
  - Read the supplementary reading for today's class (see previous slide)
- **While attempting the assignment, you are encouraged to:**
  - Google (G)/ChatGPT/ChatISA any R that you need.
  - Examine any R functions by utilizing on its help document using the `?function_name`