ISA 401: Business Intelligence & Data Visualization

03: Importing and Exporting Data

Fadel M. Megahed, PhD

Endres Associate Professor
Farmer School of Business
Miami University

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

Fall 2023

Quick Refresher from Last Class

✅ Describe how and why we use scripted languages in this course.

✅ Utilize the project workflow in RStudio (we will try to use that as an IDE for and).

✅ Understand the syntax, data structures and functions in both and .

✅ Understand the potential impact of LLMs on businesses and explore how they can be leveraged in the context of this class.

Learning Objectives for Today's Class

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

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)]`.

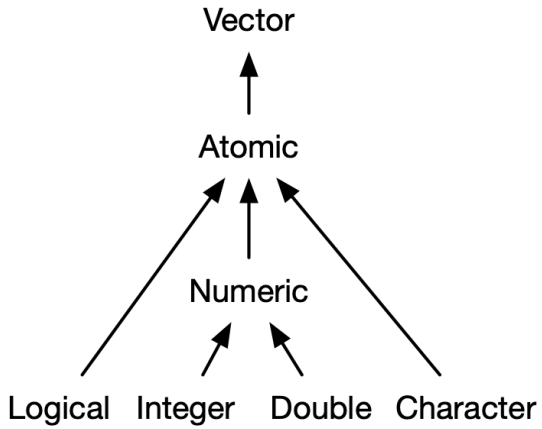
```
x_vec = rnorm(3)
x_vec
```

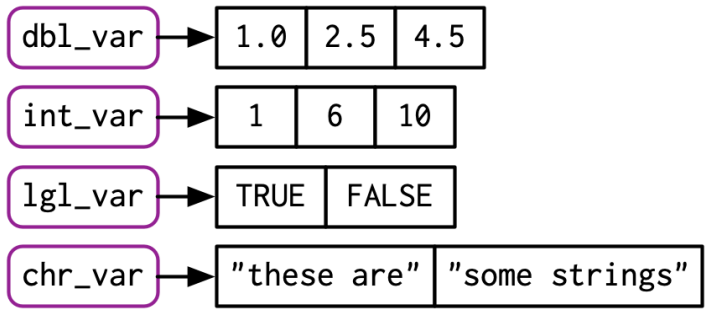
```
## [1] -2.0234707 -0.3801819  1.0253006
```

```
x_vec[2]
```

```
## [1] -0.3801819
```

```
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 `[]`

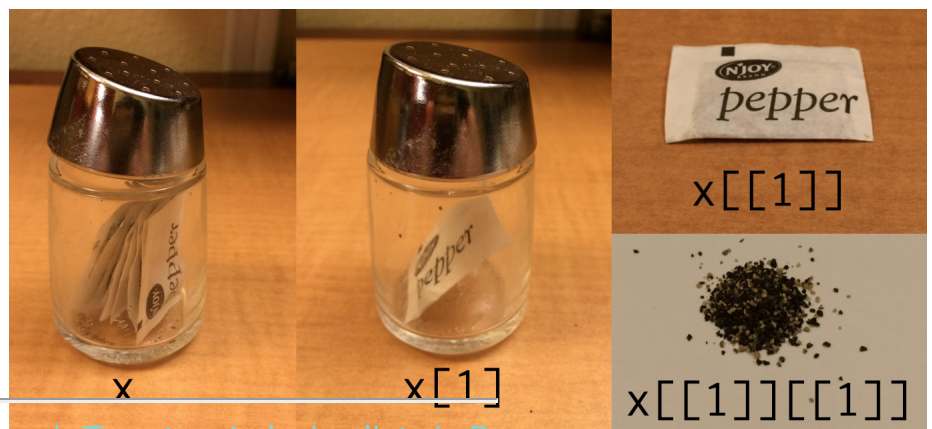
```
lst[4]
```

```
## [[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,]    5    7
## [2,]   10    9
## [1] "-----------------"
## [1] 7
```

A 3×4 character matrix

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

```
##      [,1] [,2] [,3] [,4]
## [1,] "v"  "i"  "w"  "g"
## [2,] "p"  "t"  "e"  "y"
## [3,] "h"  "m"  "u"  "c"
```

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

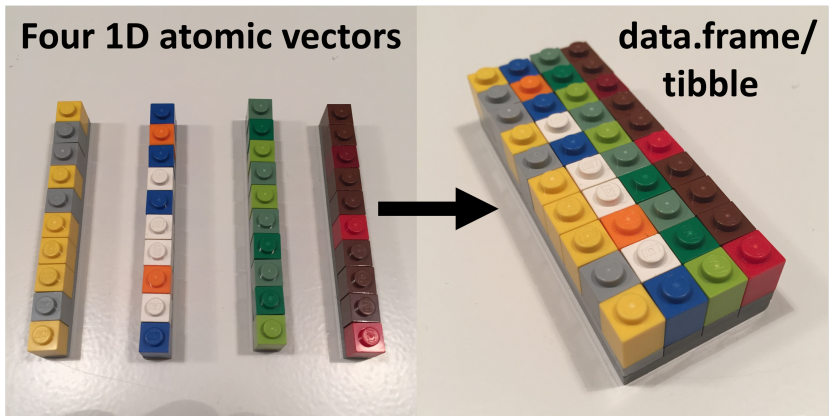
```
##      [,1] [,2]
## [1,] "i"  "w"
```

Tibbles

```
library(tibble)

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

fsb_tbl <- 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.

Translating R Data Structures to Python

R Data Structure	Description	R Subsetting (Multiple Methods)	Python Equivalent	Python Subsetting (Multiple Methods)
Vector	1D array, single type	vector[index] vector[c(1,2)] vector[-1]	List (with single type) or NumPy array	list[-1] list[1:3] array[1:3]
Matrix	2D array, single type	matrix[row, col] matrix[1,] matrix[,1]	2D List (with single type) or 2D NumPy array	list[row][col] array[row, col] array[row,:] array[:,col]
Data Frame	2D table, multiple types	df[row, col] df[1,] df[, "col"] df$col	Pandas DataFrame	df.loc[row, col] df.iloc[row, col]
List	Ordered collection, multiple types	list[[index]] list$element_name list[[1]][1]	List	list[index] list[index][subindex]
Dictionary	Key-value pairs	list$element_name	Dictionary	dict[key] dict.get(key)

Note: Indexing starts at **0** in Python and at **1** in R.

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()`
  - In , you can either use `pandas.read_table(filepath)` or `pd.read_csv(fpath, sep='\t')`.

Some Details on Reading CSV Data Files

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


```
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 R and Python

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

- Import CSV files into your 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 examples in class:
      - **WH Visitors Log**: https://www.whitehouse.gov/disclosures/visitor-logs/
      - **FRED Data:** e.g., Unempolyment Rate (UNRATE)
    - **GitHub** Repositories, e.g.,
      - NYT COVID-19 Repo - by state COVID-19 data
      - Women's Rights Around the World - focusing on `WomenTotal.csv`

05:00

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

05:00

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 5 minutes, use an R/Python 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 .....

05:00

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 R haven Python can be used to read and write formats used by other statistical packages. Example functions include:

- SAS
  - `.sas7bdat` with `read_sas()` | In Python, `pd.read_sas()`
- Stata
  - `.dta` with `read_dta()` | In Python, `pd.read_stata()`
- SPSS
  - `.sav` with `read_sav()` In Python, `pd.read_spsss()`

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. In , we will use `pandas.read_json()`

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?**

Please note the following from the demo:

- How the `pd.read_()` has a consistent approach to data importing?

Data export 

From Read to Write

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

Do these functions come from the same package?

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

`read_*()` to `DataFrame.to_*()`

Do these functions come from the same package?

```
import pandas as pd

covid_df_py = pd.read_csv('https://github.com/nytimes/covid-19-data/raw/master/us-states.csv

covid_df_py.to_excel('./data/covid_example.xlsx')
```

Recap

Summary of Main Points

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

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

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 highlighted in Class 01 Slide 39
- **While attempting the assignment, you are encouraged to:**
  - Google (G) any R or Python that you need.
  - Consult the readr cheatsheet and the pandas cheatsheet to quickly reference common use cases.