ISA 401: Business Intelligence & Data Visualization

02: Introduction to R

Fadel M. Megahed, PhD

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

Twitter: FadelMegahed
GitHub: fmegahed
Email: fmegahed@miamioh.edu
Office Hours: Automated Scheduler for Virtual Office Hours

Spring 2022

Quick Refresher from Last Class

✅ Describe course objectives & structure

✅ Define data visualization & describe its main goals

❌ Describe the BI methodology and major concepts

Learning Objectives for Today's 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 first R script.

Pedagogy Behind Using R in this Course [1]

```
crashes =
  # reading the data directly from the source
  read_csv("https://data.cincinnati-oh.gov/api/views/rvmt-pkmq/rows.csv?accessType=DOWNLOAD"
  # changing all variable names to snake_case
  clean_names() %>%
  # selecting variables of interest
  select(address_x, latitude_x, longitude_x, cpd_neighborhood, datecrashreported, instanceid
  # engineering some features from the data
  mutate(
    datetime = parse_date_time(datecrashreported, orders = "'%m/%d/%Y %I:%M:%S %p", tz =  'A
    hour = hour(datetime),
    date = as_date(datetime)   )
```

The Beauty of Programming Languages

- Programming languages are **languages**.
- **It's just text** -- which gives you access to **two extremely powerful techniques**!!!

Source: Content in "The Beauty of Programming Languages" is from Hadley Wickham's You Can't Do Data Science in a GUI

Pedagogy Behind Using R in this Course [2]

The Beauty of Programming Languages (Continued)

- `Ctrl` + `C`
- `Ctrl` + `V`
- In addition, programming languages are generally
  - Readable (IMO way easier than trying to figure what someone did in an)
  - Open (so you can it)
  - Reusable and reproducible (so you can reuse your code for similar problems and other people can get the same results as you easily)
  - Diffable (version control is extremely powerful)

Source: The bullets in "The Beauty of Programming Languages" are based on Hadley Wickham's You Can't Do Data Science in a GUI

Pedagogy Behind Using R in this Course [3]

Why R Specifically?

- It is a general-purpose programming language, which originated for statistical analysis.
- As of Jan 25, 2022, there are 18,766 packages on CRAN (not including those on GitHub).
- The `tidyverse` group of packages have simplified the workflow for data analytics/science.

What R can do?

fun stuff

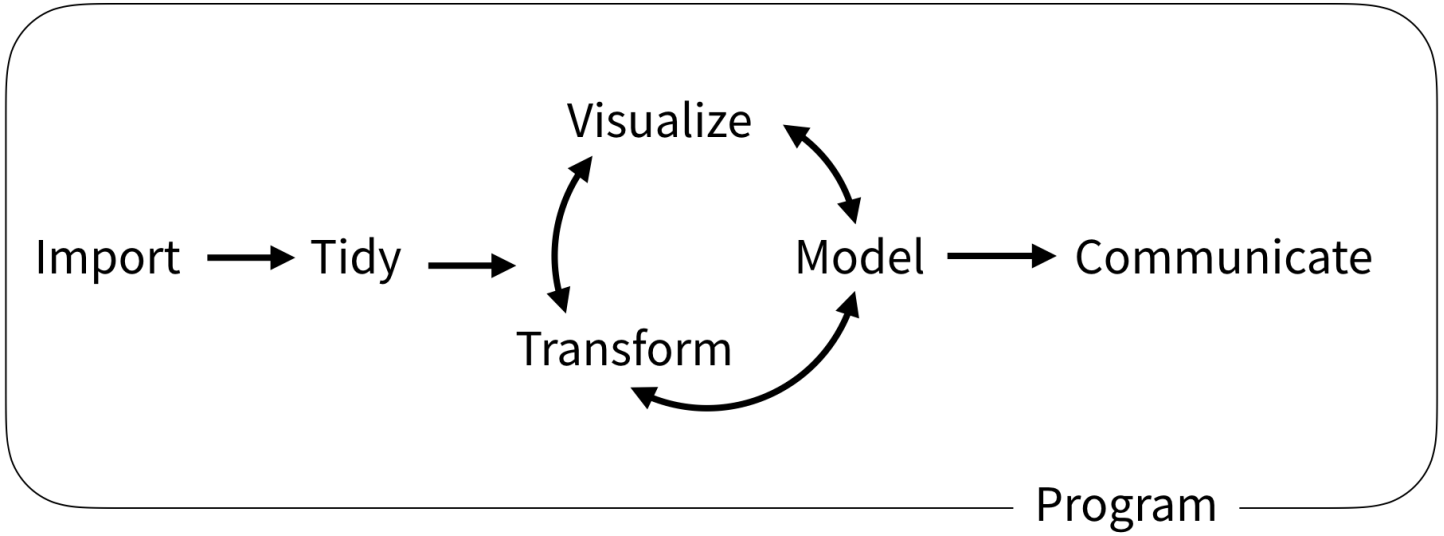
{memer} for generating memes in R

```
meme_get("SuccessKid") %>%
  meme_text_top("I WAS WORRIED \n ABOUT ISA 401") %>%
  meme_text_bottom("But IT WILL BE A \n GREAT LEARNING EXPERIENCE")
```

What R can do?

{tidyverse} for simplying the analytics workflow

data analytics

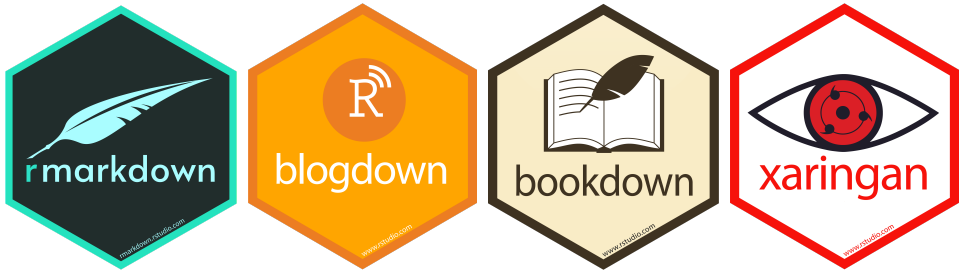


Source: Slide is based on Earo Wang's STAT 220 Slides

What R can do?

communication

R Markdown



- {rmarkdown} for reproducible analysis (e.g., see Analysis of COVID-19 Cases, Fatigue Analysis, and ISA Overview)
- {blogdown} for blogs
- {bookdown} for books
- {xaringan} for slides (401 slides!)

R Markdown documents are fully reproducible: weaving narrative text and code together.

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

What R can do?

communication

{flexdashboard} for creating Quick Dashboards

- flexdashboard is an R package that makes it easy to build interactive web apps straight from R Markdown.

How to Learn R (Any Programming Language)

- 🗣 **Get hands dirty**!!
- 📖 Documentation! Documentation! Documentation!
- 🔎 (Not surprisingly) Learn to Google: what that error message means (I G a lot 😆)

Source: Slide is based on Kia Ora's How I Learn a Technology.

The RStudio Interface, Setup and a Project-Oriented Workflow for your Analysis

RStudio®

> If R were an airplane, RStudio would be the airport, providing many, many supporting services that make it easier for you, the pilot, to take off and go to awesome places. Sure, you can fly an airplane without an airport, but having those runways and supporting infrastructure is a game-changer.
>
> -- Julie Lowndes

RStudio interface

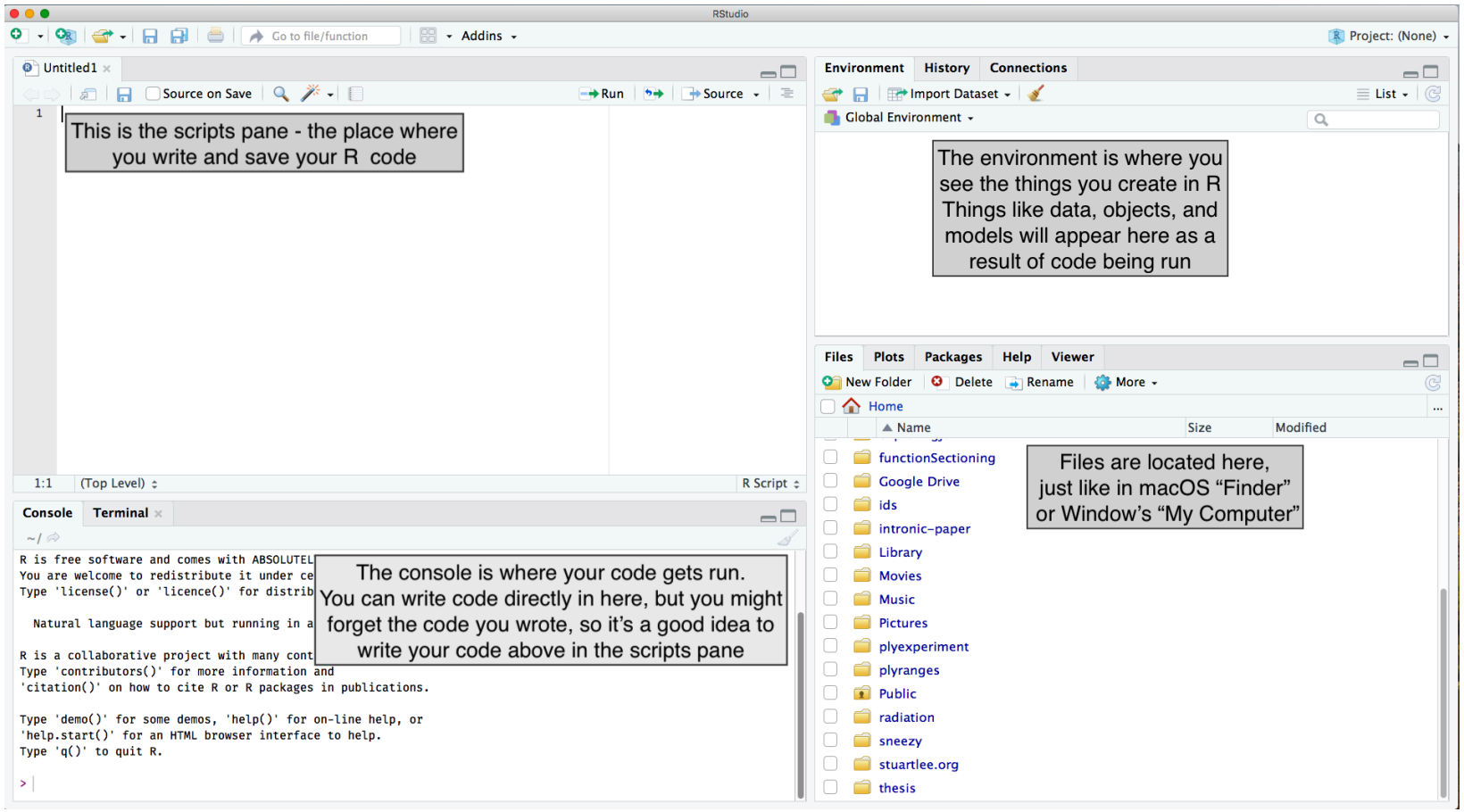


image credit: Stuart Lee

Setting up RStudio (do this once)

Go to **Tools** > **Global Options**:

Uncheck `Workspace` and `History`, which helps to keep R working environment fresh and clean every time you switch between projects.

Your Turn

02:00

> Over the next 2 minutes, change the RStudio appearance up to your taste

What is a project?

- Each university course is a project, and get your work organised.
- A self-contained project is a folder that contains all relevant files, for example my `ISA 401/` 📂 includes:
  - `Spring2022.Rproj`
  - `Lectures/`
    - `01_Introduction/`
      - `01-introduction.Rmd`, `figures/`, `data/`, etc.
    - `02_Introduction_to_R/`
      - `01-Introduction_to_R.Rmd`, `figures/`, `data/`, etc.
- All working files are **relative** to the **project root** (i.e. `isa401/`).
- The project should just work on a different computer.

STOP DOING THIS!

If this is not your first `fontawesome::fa("r-proj")`, you have probably seen some variation of the following as your first couple of lines in a `fontawesome::fa("r-proj")` script file:

```
setwd("C:\Users\jenny\path\that\only\I\have")
rm(list = ls())
```

Jenny Bryan (an RStudio software engineer and a highly respected `#rstats` developer) will set your computer on fire 🔥 if you have these lines in your script file.

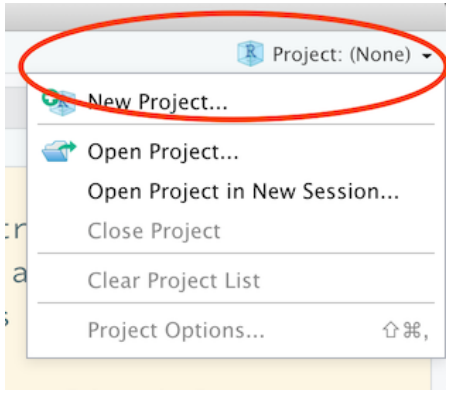
Main reason: If you are using an .Rproj and did setup the global options as illustrated earlier, you no longer need these steps for any of your code (as long as you see the project icon on the top-right of your screen).

For a more detailed discussion, please read her blog post on workflow vs script.

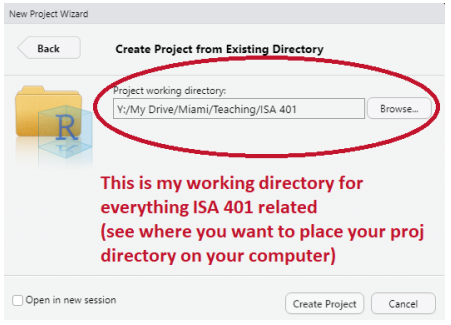
Create an RStudio project `.Rproj` for Our Course

02:00

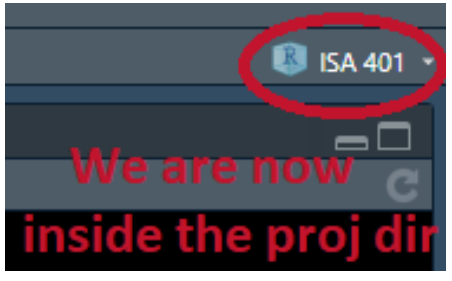
- Click the **Project** icon on the top right corner



- **New Directory**/**Existing Directory** > **New Project** > **Create Project**



- Open the project



101: Operators

Assignment

R has three assignment operators: <-, =, and ->, which can be used as follows.

```
x1 <- 5

x2 = 5

5 -> x3

print(paste0("The values of x1, x2, and x3 are ", x1, ", ", x2, ", and ", x3, " respectively
```

```
## [1] "The values of x1, x2, and x3 are 5, 5, and 5 respectively"
```

The operator <- can be written using the shortcut `Alt` + `-` on a Windows machine. If we focus on the first two operators, you would notice that the assignment consists of three parts:

- The left-hand side: **variable names** (`x1` or `x2`),
- The assignment operator: `<-` (or alternatively `=`), and

Retrieval

We can retrieve/call the object using its name as follows:

```
x1
```

```
## [1] 5
```

```
x3
```

```
## [1] 5
```

Retrieval: Three Common Errors

Case issue: object names in R are **case sensitive**.

```
X1 # should be x1 instead of X1 (see last slide)
```

```
## Error in eval(expr, envir, enclos): object 'X1' not found
```

Typo: A spelling error of some sort

```
y3 # should be x3 instead of y3 (see last slide)
```

```
## Error in eval(expr, envir, enclos): object 'y3' not found
```

Object not saved: e.g., you clicked **Enter** instead of **Ctrl + Enter** when running your code

```
rm(x2) # removing x2 from the global environment to mimic error
x2 # x2 is not in the global environment (see environment)
```

```
## Error in eval(expr, envir, enclos): object 'x2' not found
```

Arthimetic Operators

While we will not specifically talk about doing math in R, the operators below are good to know.

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Source: The above table is scraped from Quick-R: Operators.

Logical Operators

Logical operators are operators that return `TRUE` and `FALSE` values.

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Source: The above table is also scraped from Quick-R: Operators.

101: Syntax, Data Types, Data Structures and Functions

Coding Style

> Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread.
> -- The tidyverse style guide

R style guide

✅ `snake_case`

❌ `camelCase` (Javascript)

❌ `PascalCase` (Python)

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

R is a Vector Language: Sample Output

Learning from a **sample of ten obs. from the standard normal distribution** (i.e., $x \sim \mathcal{N}(0, 1)$)

```
x_vec = rnorm(n=10, mean = 0, sd = 1) # generating std normal dist data
x_vec > 0 # finding which elements in x are larger than 0
```

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

```
sum(x_vec > 0) # summing the number of elements (i.e., how many are > 0)
```

```
## [1] 4
```

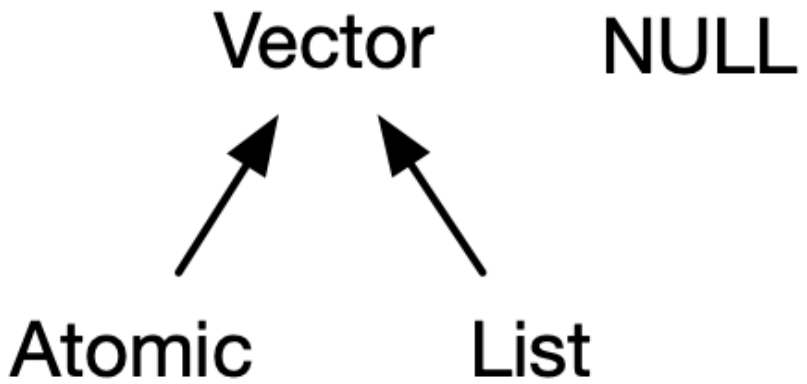
If we focus on the obtained outputs, we can see that **both** are vectors:

- `x_vec > 0` returns a vector of size 10 (TRUE/FALSE for each element)
- `sum(x_vec > 0)` returns a vector of size 1

There are no scalars in R 🤯!!!

R is a Vector Language: Types and Attributes

- **Vectors** come in **two flavors**, which differ by their **elements' types:**
  - **atomic vectors --** all elements **must have the same type**
  - **lists --** elements **can** be different
- Vector have two important **attributes:**
  - **Dimension** turns vectors into matrices and arrays, checked using `dim(object_name)`.
  - The **class** attribute powers the S3 object system, checked using `class(object_name)`.

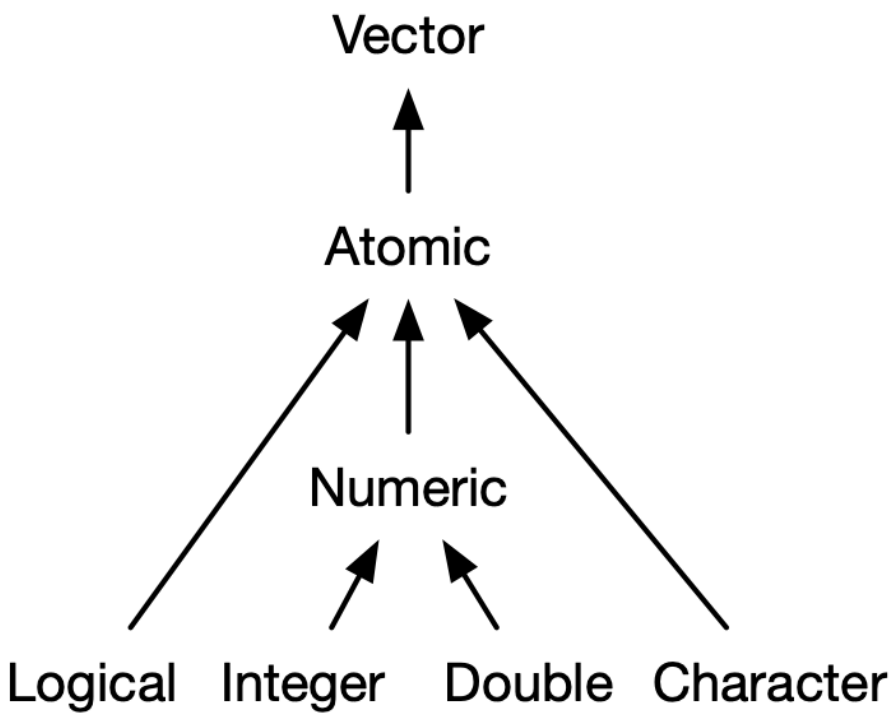



```
class(x_vec)
```

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

Source: The content and image are from Hadley Wickham's Advanced R: Chapter 3 on Vectors

R is a Vector Language: Atomic Vectors



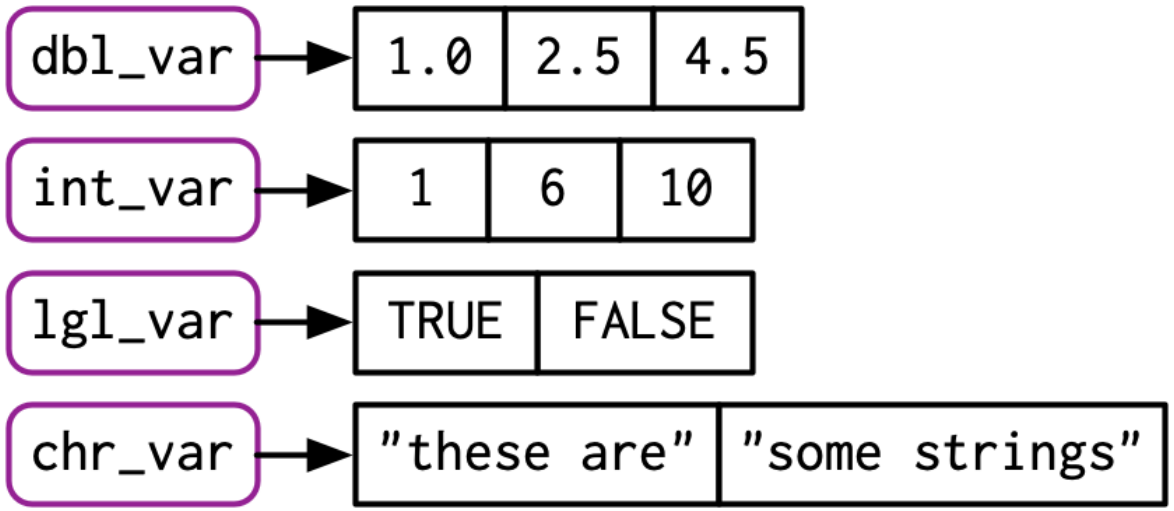

```
dim(x_vec)
```

```
## NULL
```

Atomic vectors have a dim of NULL, which distinguishes it from 1D arrays 😲!!!

Source: The image is from Hadley Wickham's Advanced R: Chapter 3 on Vectors

Data Types: A Visual Introduction [1]



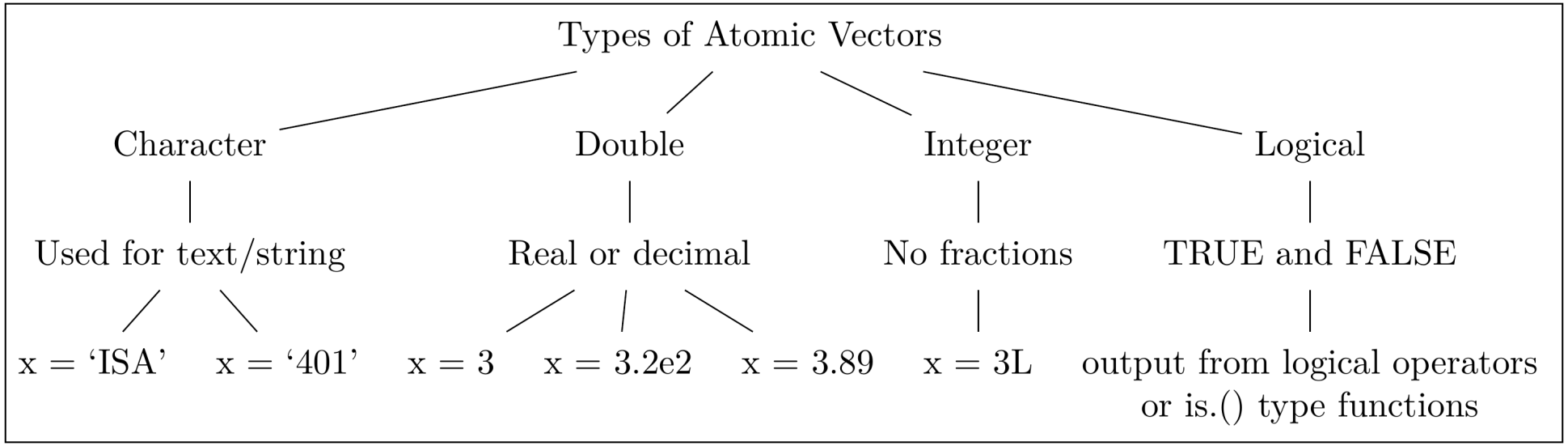
- To check the **type of** an object in R, you can use the function `typeof`:

```
typeof(x_vec)
```

```
## [1] "double"
```

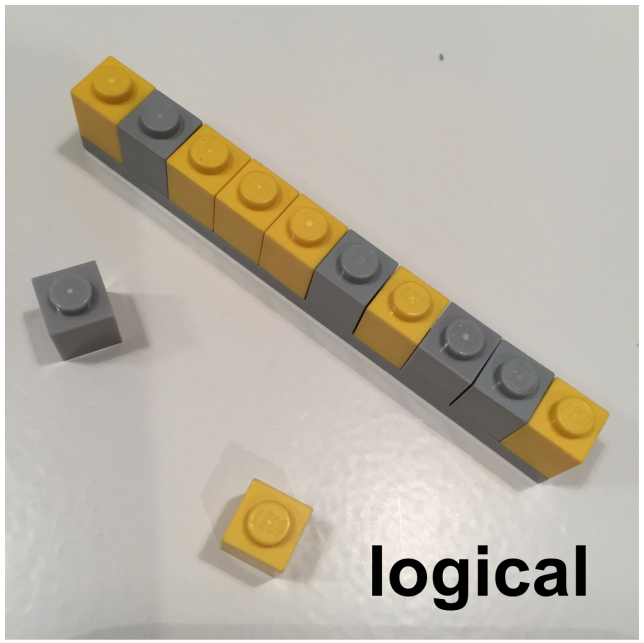
Source: The image is from Hadley Wickham's Advanced R: Chapter 3 on Vectors

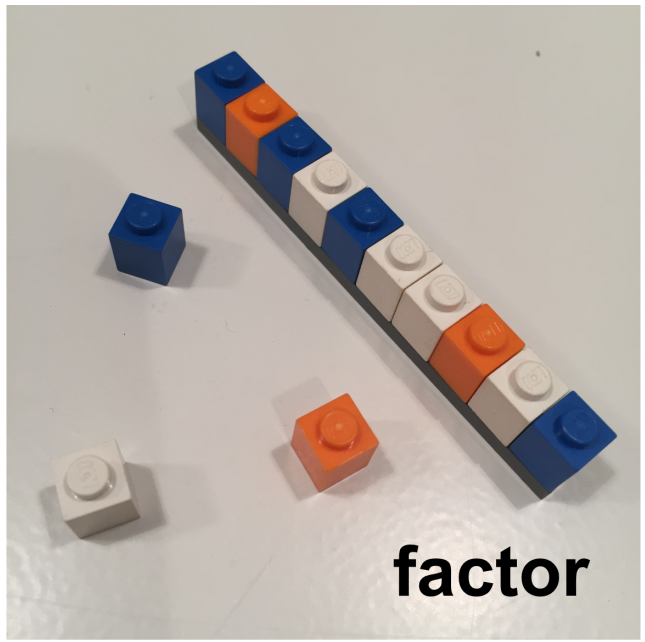
Data Types: A Visual Introduction [2]

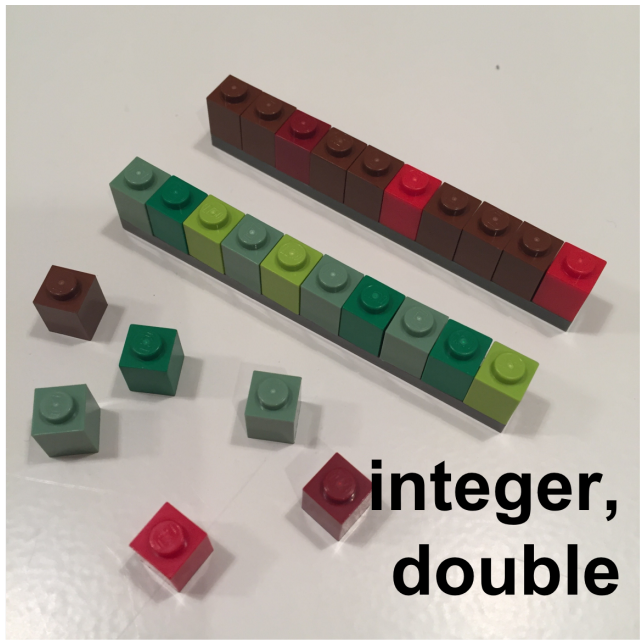


The four data types that we will utilize the most in our course.

Data Types: A Visual Introduction [3]







A visual representation of different types of atomic vectors

Source: The images are from the excellent lego-rstats GitHub Repository by Jenny Bryan

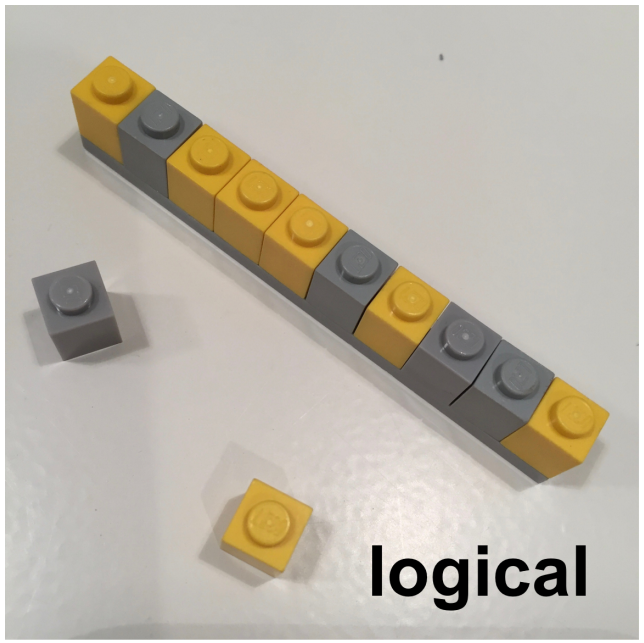
Data Types: Formal Definitions

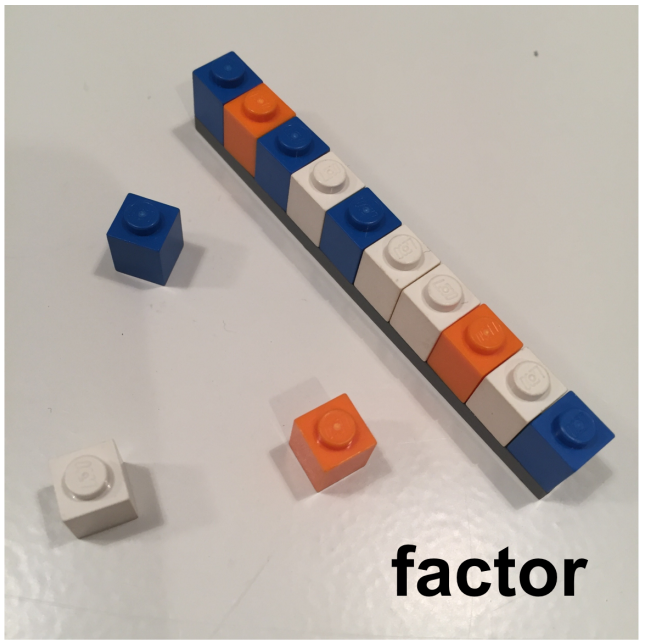
Each of the four primary types has a special syntax to create an individual value:

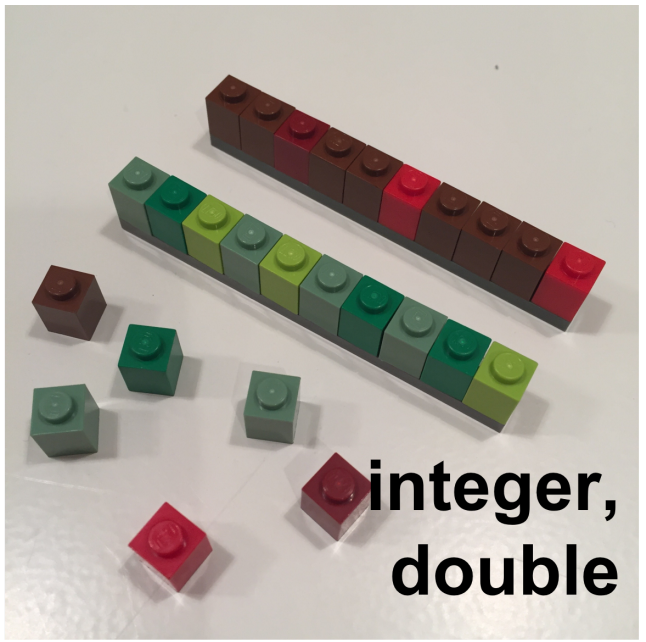
- Logicals can be written in full (`TRUE` or `FALSE`), or abbreviated (`T` or `F`).
- Doubles can be specified in decimal (`0.1234`), scientific (`1.23e4`), or hexadecimal (`0xcafe`) form.
  - There are three special values unique to doubles: `Inf`, `-Inf`, and `NaN` (not a number).
  - These are special values defined by the floating point standard.
- Integers are written similarly to doubles but must be followed by `L`(`1234L`, `1e4L`, or `0xcafeL`), and can not contain fractional values.
- Strings are surrounded by `"` (e.g., `"hi"`) or `'` (e.g., `'bye'`). Special characters are escaped with `\` see `?Quotes` for full details.

Source: The content of this slide is verbatim from Hadley Wickham's Advanced R: Chapter 3 on Vectors

Data Structures: Atomic Vector (1D)





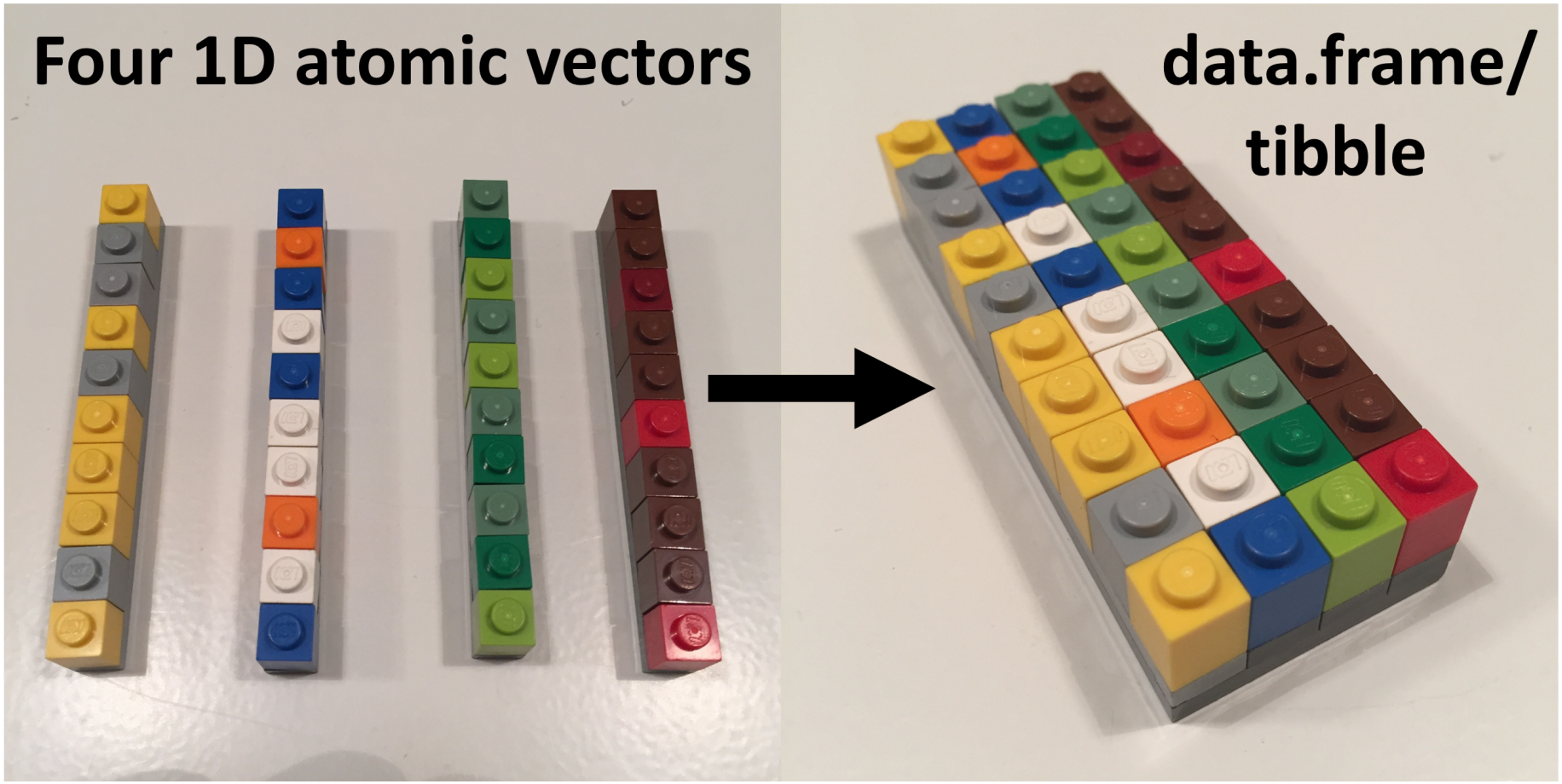


Keeping the visual representation of different types of atomic vectors in your head!!

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

Source: The images are from the excellent lego-rstats GitHub Repository by Jenny Bryan

Data Structures: 1D → 2D [Visually]



Source: The images are from the excellent lego-rstats GitHub Repository by Jenny Bryan

Data Structures: 1D ➡ 2D [In Code]

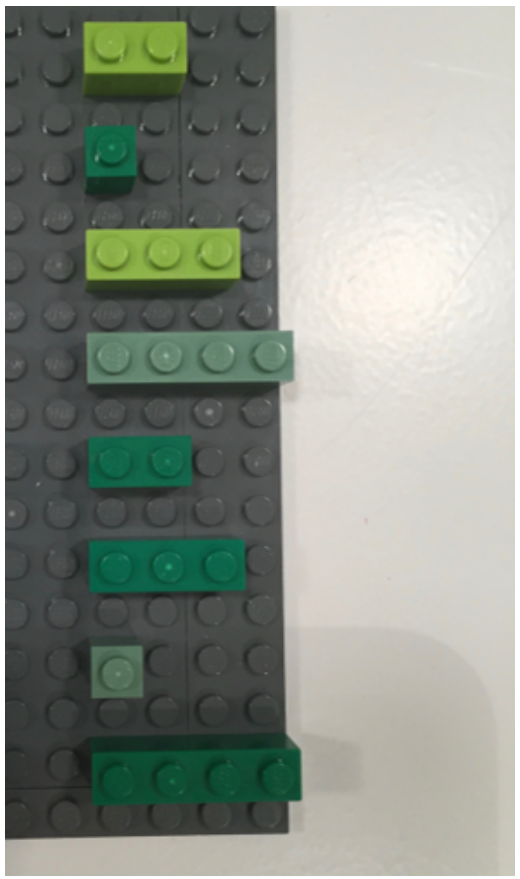
```
library(tibble)

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

```
## # A tibble: 5 x 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
```

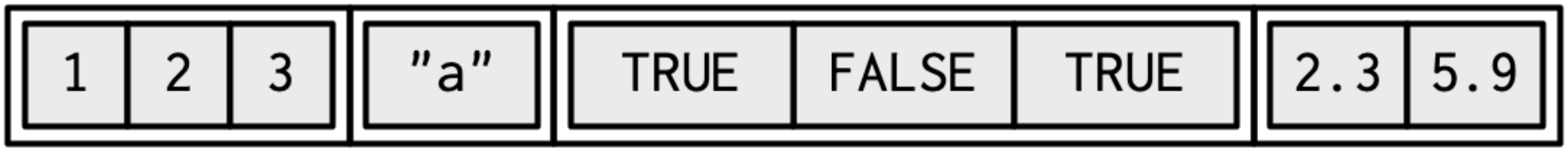
Data Structures: Lists [1]

An object contains elements of **different data types**.



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

Data Structures: Lists [2]




```
lst <- list( # list constructor/creator
  1:3, # atomic double/numeric vector  of length = 3
  "a", # atomic character vector of length = 1 (aka scalar)
  c(TRUE, FALSE, TRUE), # atomic logical vector of length = 3
  c(2.3, 5.9) # atomic double/numeric vector of length =3
)
lst # printing the list
```

```
## [1] "1:3"                 "a"                   "c(TRUE, FALSE, TRUE)"
## [4] "c(2.3, 5.9)"
```

Source: Image is from Hadley Wickham's Advanced R: Chapter 3 on Vectors

Data Structures: Lists [3]

data type

```
typeof(lst) # primitive type
```

```
## [1] "list"
```

data class

```
class(lst) # type + attributes
```

```
## [1] "list"
```

data structure

```
str(lst)
# sublists can be of diff lengths and typ
```

```
## List of 4
##  $ : int [1:3] 1 2 3
##  $ : chr "a"
##  $ : logi [1:3] TRUE FALSE TRUE
##  $ : num [1:2] 2.3 5.9
```

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

Data Structures: Lists [3]

A list can contain other lists, i.e. **recursive**

```
# a named list
str(
  list(first_el = lst, second_el = iris)
  )
```

```
## List of 2
##  $ first_el :List of 4
##   ..$ : int [1:3] 1 2 3
##   ..$ : chr "a"
##   ..$ : logi [1:3] TRUE FALSE TRUE
##   ..$ : num [1:2] 2.3 5.9
##  $ second_el:'data.frame':    150 obs. of  5 variables:
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..$ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##   ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Data Structures: Lists [4]

Subset by `[]`

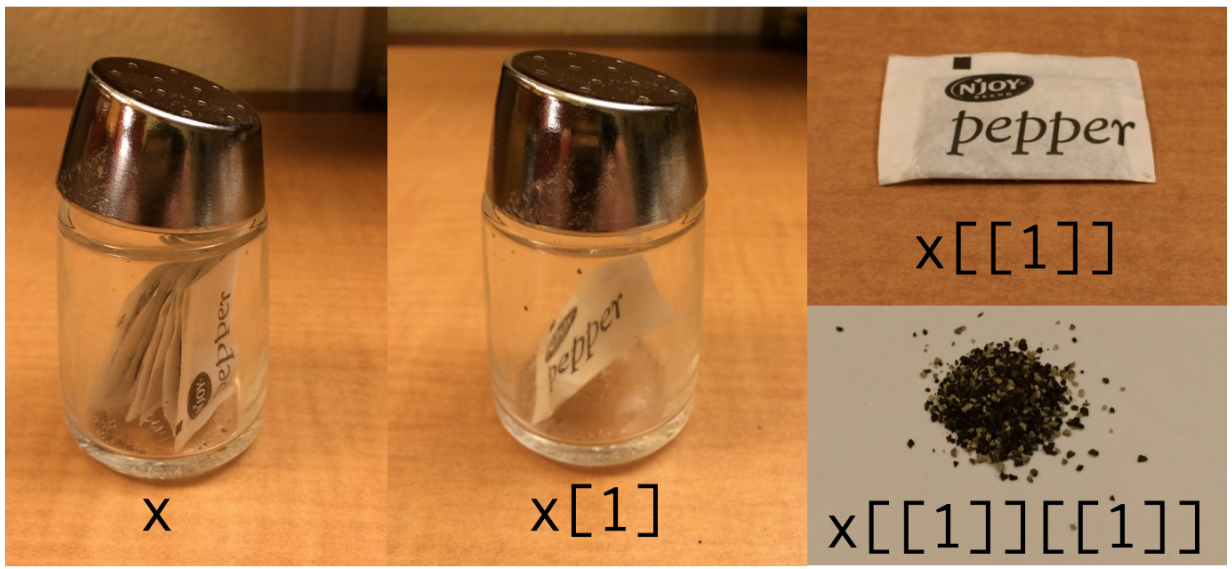
```
lst[1]
```

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

Subset by `[[]]`

```
lst[[1]]
```

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



Sources: The slide is based on Earo Wang's STAT 220 slides and image is from Hadley Wickham's Tweet on Indexing lists in R.

Data Structures: Matrices

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
str(x_mat) # its structure?
```

```
##  int [1:2, 1:2] 1 7 2 3
```

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

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    7    3
## [1] "-----------------"
```

A 3×4 character matrix

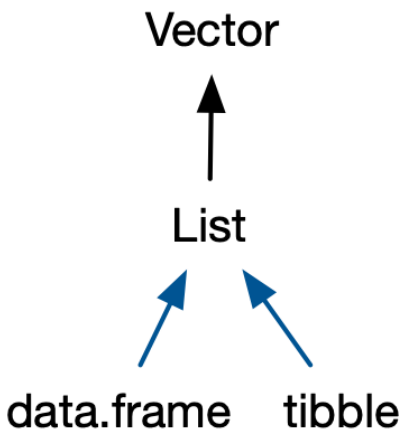
```
x_char = matrix(
  sample(letters, size = 12), nrow = 3, n
x_char
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "b"  "e"  "v"  "f"
## [2,] "n"  "m"  "g"  "t"
## [3,] "y"  "d"  "j"  "h"
```

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

```
##      [,1] [,2]
## [1,] "e"  "v"
```

Data Structures: Data Frames [1]



> If you do data analysis in R, you're going to be using data frames. A data frame is a named list of vectors with attributes for `(column) names`, `row.names`, and its class, "data.frame". -- Hadley Wickham

Source: Image is from Hadley Wickham's Advanced R: Chapter 3 on Vectors

Data Structures: Data Frames [2]

```
df1 <- data.frame(x = 1:3, y = letters[1:3])

typeof(df1) # showing that its a special case of a list
```

```
## [1] "list"
```

```
attributes(df1) # but also is of class data.frame
```

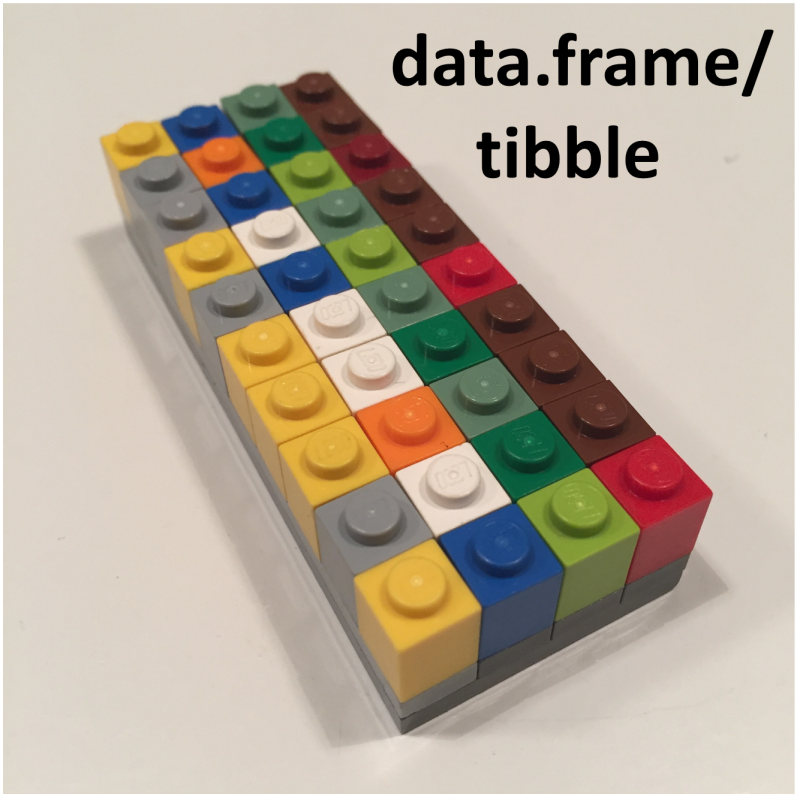
```
## $names
## [1] "x" "y"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3
```

In contrast to a regular list, a data frame has **an additional constraint: the length of each of its vectors must be the same.** This gives data frames their **rectangular structure.**

Source: Content is from Hadley Wickham's Advanced R: Chapter 3 on Vectors

Data Structures: Data Frames [3]

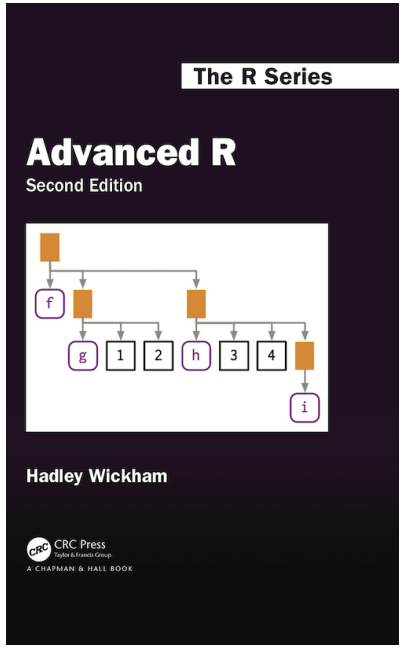
As noted in the creation of `df1`, columns in a data frame can be of different types. Hence, it is more widely used in data analysis than matrices.



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

Data Structures: So What is a Tibble Anyway?

> Tibble is a **modern reimagining of the data frame**. Tibbles are designed to be (as much as possible) **drop-in replacements for data frames** that fix those frustrations. A concise, and fun, way to summarise the main differences is that tibbles are **lazy and surly: they do less and complain more**. -- Hadley Wickham



To learn more about the basics of tibble, please consult the reference below:

- Data frames and tibbles (Click and read from 3.6 up to and including 3.6.5)

Functions

A function call consists of the **function name** followed by one or more **argument** within parentheses.

```
temp_high_forecast = c(37, 37, 26, 22, 37, 27, 29, 40)
mean(x = temp_high_forecast)
```

```
## [1] 31.875
```

- function name: `mean()`, a built-in R function to compute mean of a vector
- argument: the first argument (LHS `x`) to specify the data (RHS `temp_high_forecast`)

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

Function Help Page

02:00

Check the function's help page with `?mean`

Class Activity

> *Please take 2 minutes to investigate the help page for `mean` in R Studio.*

```
mean(x = temp_high_forecast, trim = 0, na.rm = FALSE, ...)
```

- Read **Usage** section
  - What arguments have default values?
- Read **Arguments** section
  - What does `trim` do?
- Run **Example** code

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

Function Arguments

Match by **positions**

```
mean(temp_high_forecast, 0.1, TRUE)
```

```
## [1] 31.875
```

Match by **names**

```
mean(x = temp_high_forecast, trim = 0.1,
```

```
## [1] 31.875
```

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

Use Functions from Packages

```
library(dplyr)
cummean(temp_high_forecast)
```

```
## [1] 37.00000 37.00000 33.33333 30.50000 31.80000 31.00000 30.71429 31.87500
```

```
first(temp_high_forecast)
```

```
## [1] 37
```

```
last(temp_high_forecast)
```

```
## [1] 40
```

Images sourced from https://www.wikihow.com/Change-a-Light-Bulb

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

Write Your Own Functions

```
# function_name <- function(arguments) {
#   function_body
# }
my_mean <- function(x, na.rm = FALSE) {
  summation <- sum(x, na.rm = na.rm)
  summation / length(x)
}

my_mean(temp_high_forecast)
```

```
## [1] 31.875
```

Demo

Your First R Script File

Refer to our in class code.

Recap

Summary of Main Points

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

- 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 first R script.