ISA 401: Business Intelligence & Data Visualization

02: A Quick Introduction to R, Python, and LLMs

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

Note: Image generated using https://ideogram.ai/. **Pros:** Currently best text-2-image generator while embedding text, and free (useful for professional logos). **Cons:** Not the best with hands.

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 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.

Scripted Languages (i.e.,  and )

Pedagogy Behind Using Scripted Languages

```
crashes =
  # reading the data directly from the source
  readr::read_csv("https://data.cincinnati-oh.gov/api/views/rvmt-pkmq/rows.csv?accessType=DO

  # changing all variable names to snake_case
  janitor::clean_names() |>

  # selecting variables of interest
  dplyr::select(address_x, latitude_x, longitude_x, cpd_neighborhood, datecrashreported, ins

  # engineering some features from the data
  dplyr::mutate(
    datetime = lubridate::parse_date_time(datecrashreported, orders = "'%m/%d/%Y %I:%M:%S %p
    hour = lubridate::hour(datetime),
    date = lubridate::as_date(datetime)
    )
```

Pedagogy Behind Using Scripted Languages

```
import pandas as pd
import datetime as dt

crashes = (
    # Reading the CSV file from a URL
    pd.read_csv('https://data.cincinnati-oh.gov/api/views/rvmt-pkmq/rows.csv?accessType=DOWN

    # Renaming all columns to be lowercase and replacing spaces with underscores
    .rename(columns={col: col.lower().replace(" ", "_") for col in pd.read_csv('https://data

    # Selecting only the columns of interest
    .loc[:, ["address_x", "latitude_x", "longitude_x", "cpd_neighborhood", "datecrashreporte

    # Adding new columns: 'datetime', 'hour', and 'date' derived from 'datecrashreported'
    .assign(
        datetime=lambda df: pd.to_datetime(df['datecrashreported'], format="%m/%d/%Y %I:%M:%
        hour=lambda df: df['datetime'].dt.hour,  # Extracting the hour from 'datetime'
        date=lambda df: df['datetime'].dt.date  # Extracting the date from 'datetime'
    )
)
```

The Beauty of Programming Languages

- Programming languages are **languages**.
- **It's just text** -- which gives you access to **two extremely powerful techniques**!!!
  - `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 it is easier to or **ChatGPT** 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: Content in "The Beauty of Programming Languages" is from Hadley Wickham's You Can't Do Data Science in a GUI

How to Learn Any Programming Language

- 🗣️ **Get hands dirty** ‼️
- 📖 Documentation! Documentation! Documentation!
- 🔍 (Not surprisingly) Learn to Google/ChatGPT: what that error message means (I do that 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 Interface

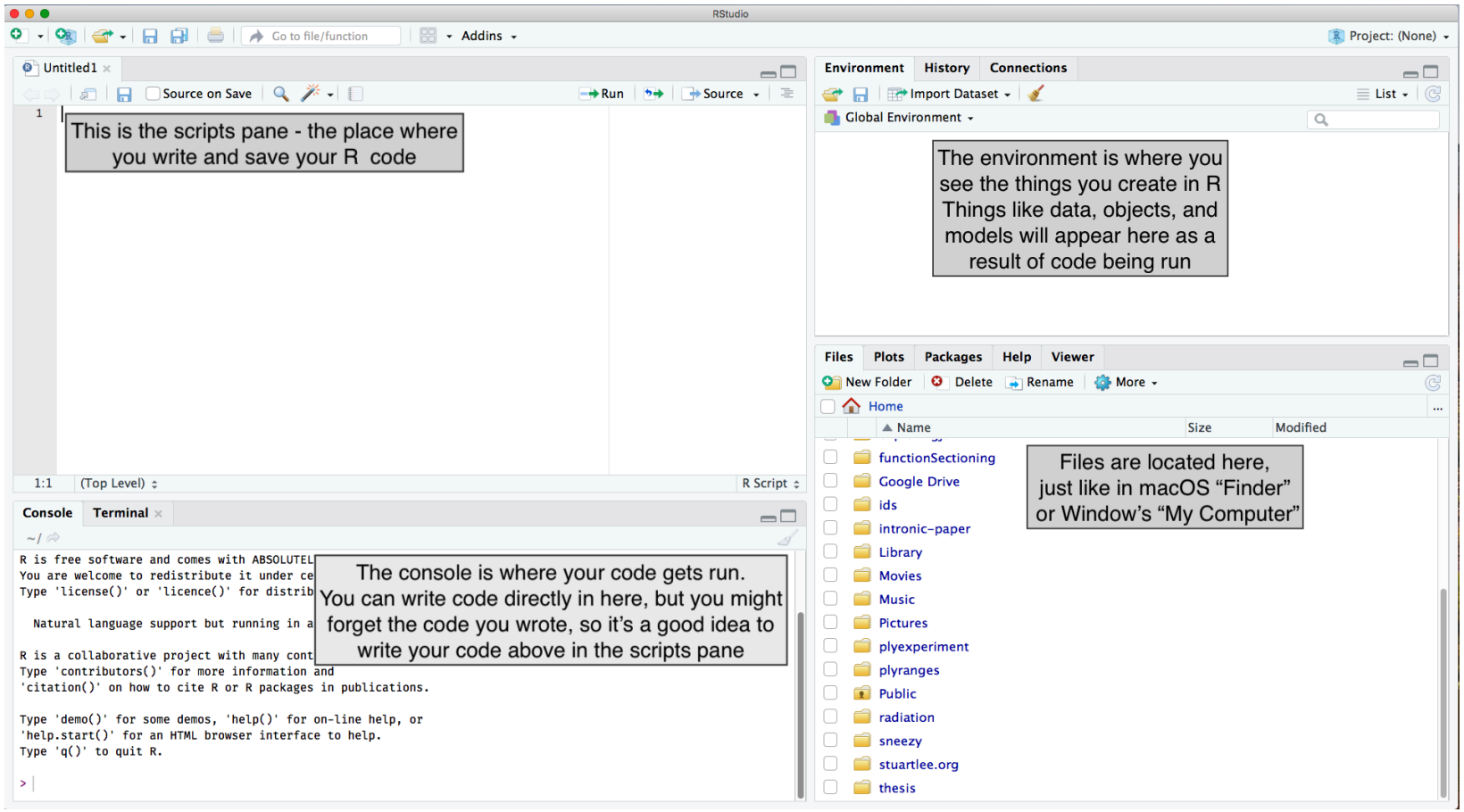
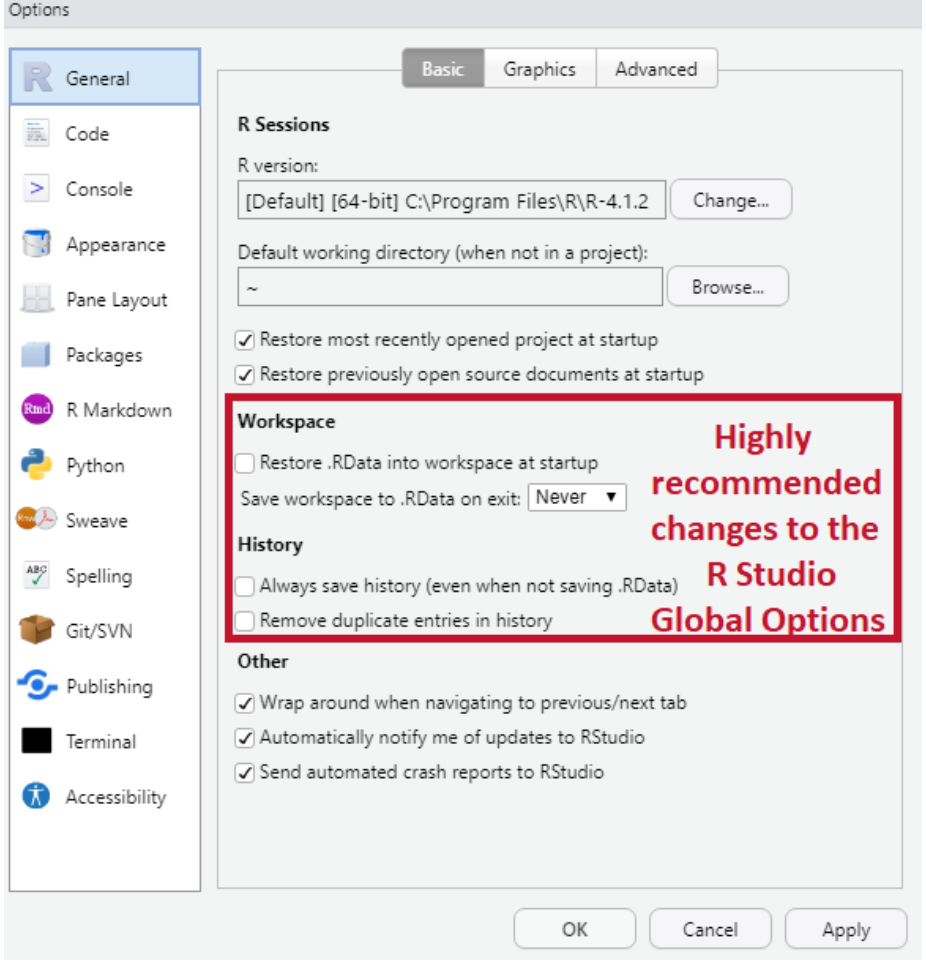


image credit: Stuart Lee

Setting up RStudio (do this once for R)

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.

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:
  - `isa401.Rproj`
  - `lectures/`
    - `01_Introduction/`
      - `01-Introduction.Rmd`, etc.
    - `02_llms_r_python/`
      - `02_llms_r_python.Rmd`, etc.
- All working files are **relative** to the **project root** (i.e. `isa401/`).
- The project should just work on a different computer (in most cases).

Setting up RStudio (do this once for)

Installing Python

- (Preferred) Install via `reticulate::install_miniconda()`.
- Install from python.org.
- Setup virtual environment and install needed

Helping R Find the Correct Version of Python

- Edit your RProfile for the project to connect to a specific version of Python,

```
Sys.setenv(RETICULATE_PYTHON =
"C:\tools\Anaconda3\envs\spc_gpt\python.exe")
```

- Configure a default version of Python to be used with RStudio via Tools -> Global Options... -> Python

Note: Feel free to use other IDEs of your choice for Python and/or R. During class, I will use RStudio for both to reduce the possible friction from using and/or setting up multiple IDEs.

Operators 101

Assignment

```r
x1 <- 5
x2 = 5
5 -> x3

print(paste0("The values of x1, x2, and x
```

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

```python
x1 = 5 # no '<-' operator
x2 = 5 # only "="
x3 = 5  # no '->' operator

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

```
## The values of x1, x2, and x3 are 5, 5,
```

The assignment consists of three parts:

- The left-hand side: **variable names** (`x1` or `x2`),
- The assignment operator: `=`, and the right-hand side: **values** (`5`)

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 and Python are **case sensitive**.

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

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

Retrieval: Three Common Errors

Case issue: object names in R and Python 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 (with a corresponding Python error message)

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

```
## name 'y3' is not defined
```

Retrieval: Three Common Errors

Case issue: object names in R and Python 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 (with a corresponding Python error message)

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

```
## name 'y3' is not defined
```

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)
```

Arthimetic Operators

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

R	Description	Python
+	addition	+
-	subtraction	-
*	multiplication	*
/	division	/
^ or **	exponentiation	**
x %% y	modulus (x mod y)	x % y
x %/% y	integer division	x // y

Logical Operators

Logical operators are operators that return `TRUE` (`True` in Python) and `FALSE` (`False` in Python) values.

R	Description	Python
<	less than	<
<=	less than or equal to	<=
>	greater than	>
>=	greater than or equal to	>=
==	exactly equal to	==
!=	not equal to	!=
!x	Not x	not x
x & y	x AND y	x & y
isTRUE(x)	test if X is TRUE	x is $True$

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

style guide

✅ `snake_case`

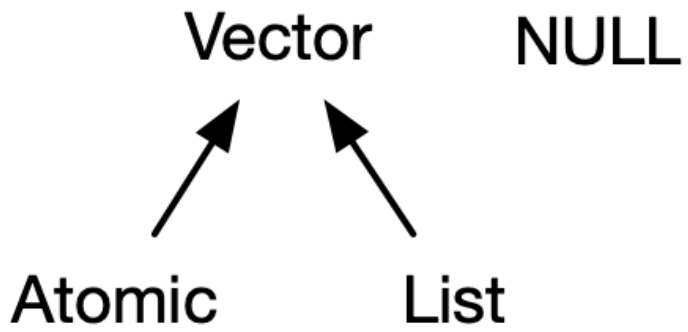
style guide

✅ `PascalCase` (Python)

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

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)`.



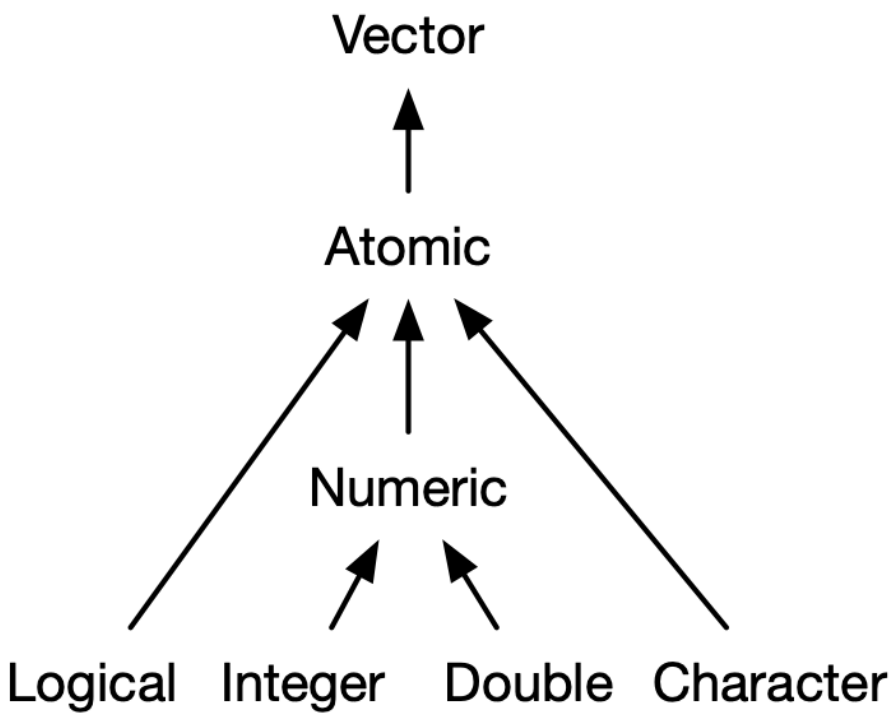

```
x_vec = rnorm(n=10, mean = 0, sd = 1)

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



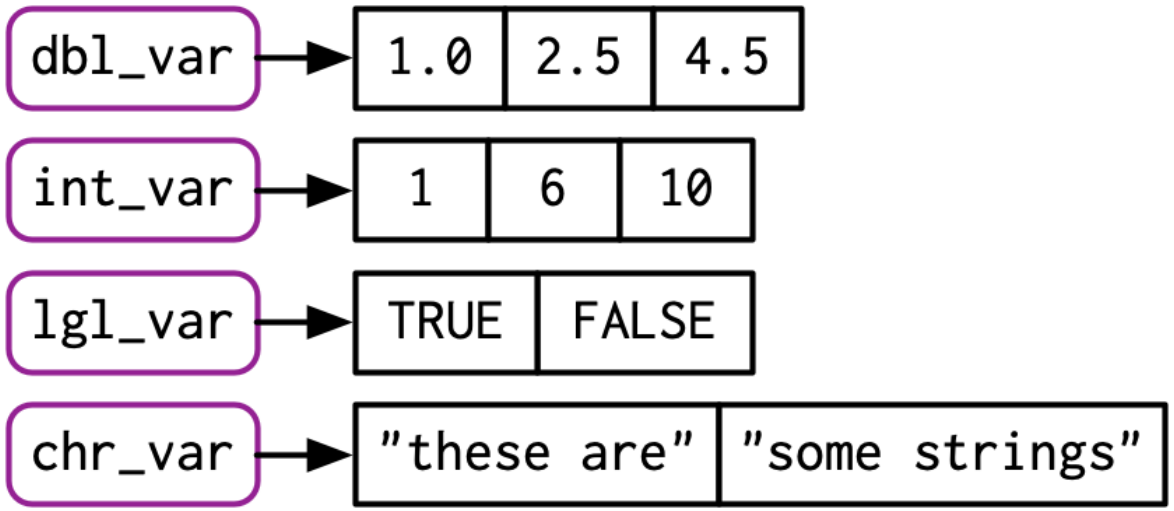

```
dim(x_vec)
```

```
## 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]



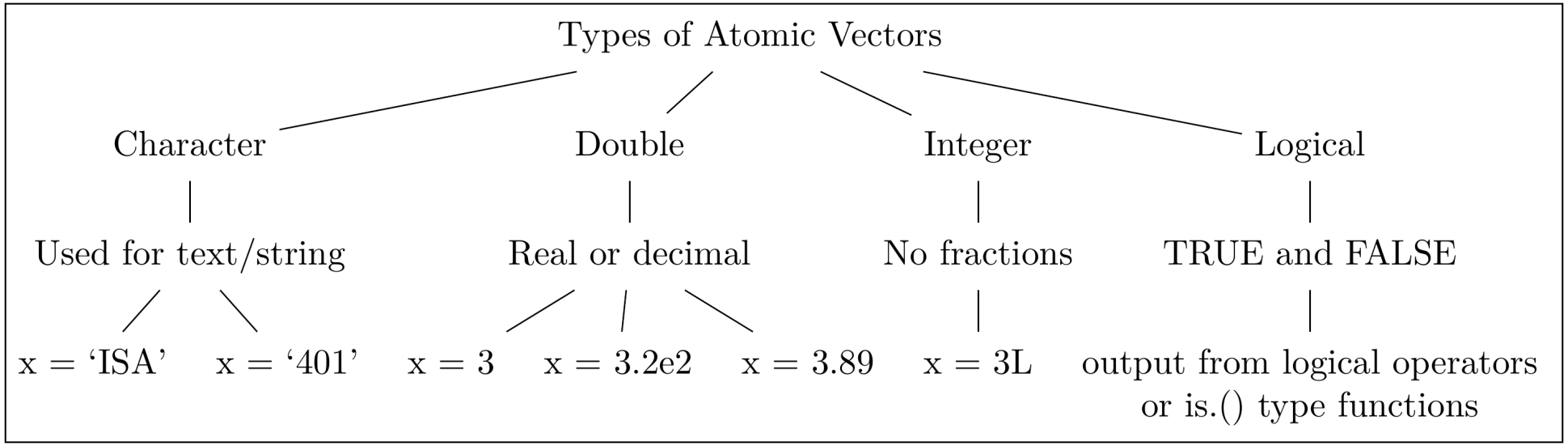
- To check the **type of** an object in R, you can use the function `typeof`:

```
typeof(x_vec)
```

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

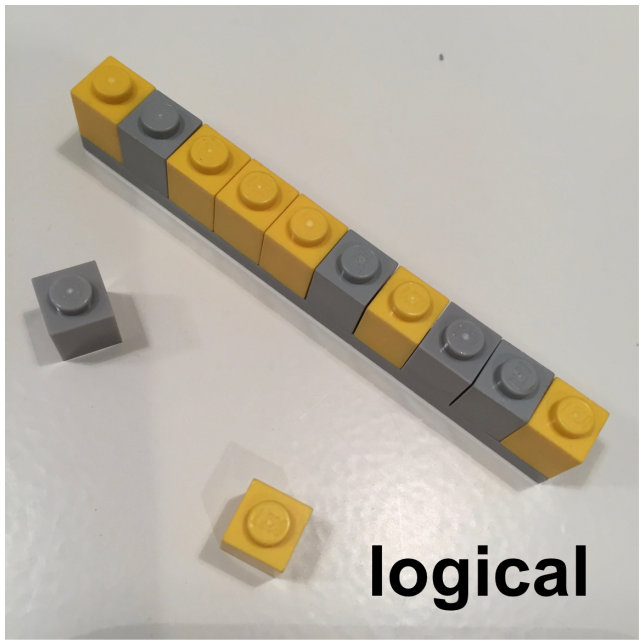
Source: The image is from Hadley Wickham's Advanced R: Chapter 3 on Vectors

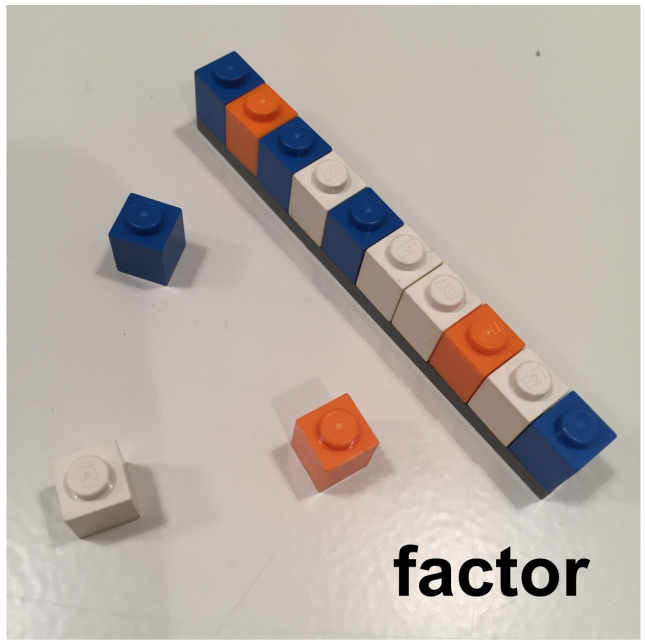
Data Types: A Visual Introduction [2]

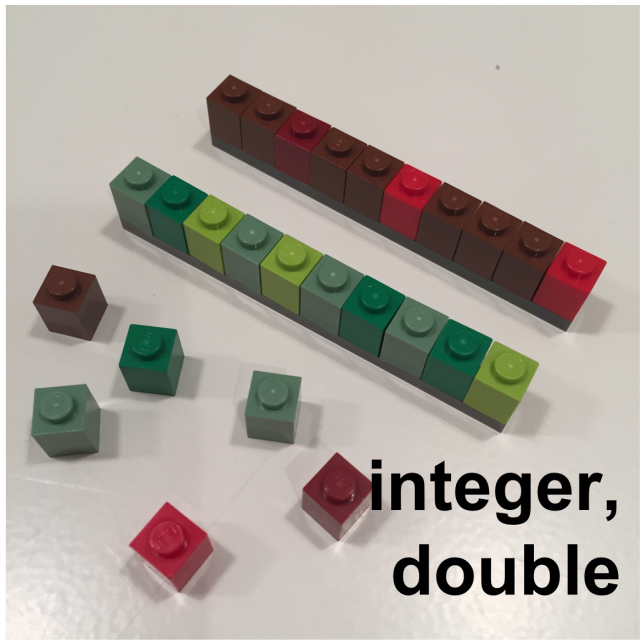


The four data types that we will utilize the most in our course.

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:

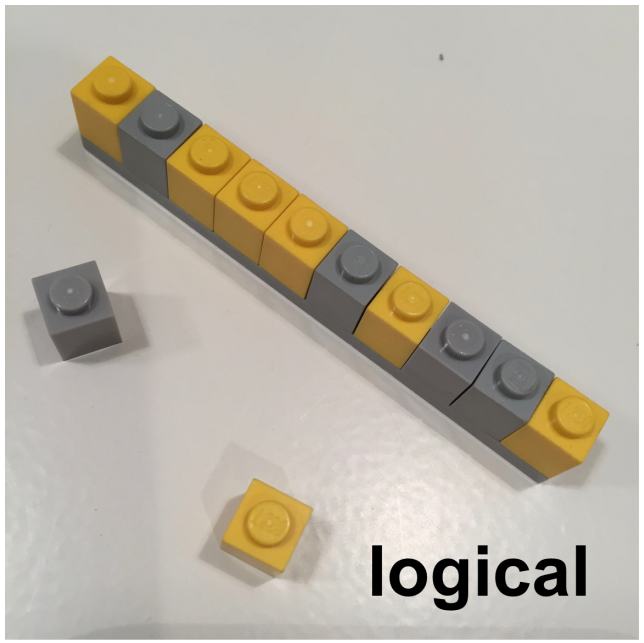
- Logicals can be written in full (`TRUE` or `FALSE`), or abbreviated (`T` or `F`).
- Doubles can be specified in decimal (`0.1234`), scientific (`1.23e4`), or hexadecimal (`0xcafe`) form.
  - There are three special values unique to doubles: `Inf`, `-Inf`, and `NaN` (not a number).
  - These are special 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.

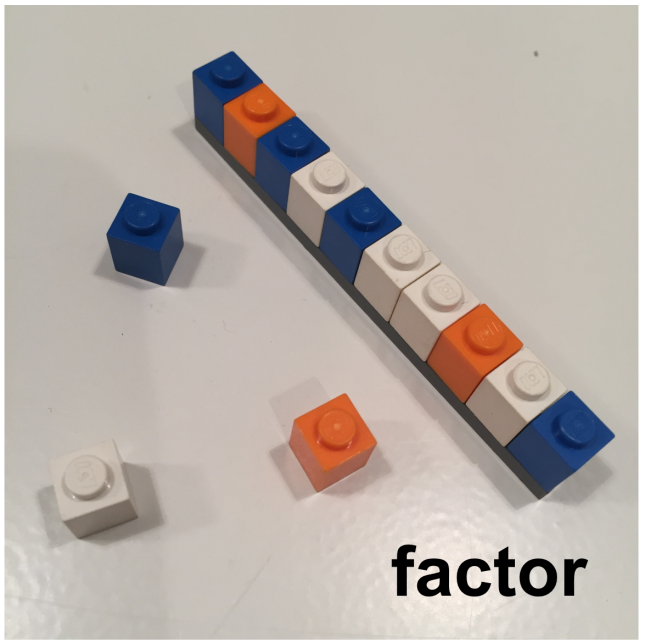
Source: The content of this slide is verbatim from Hadley Wickham's Advanced R: Chapter 3 on Vectors

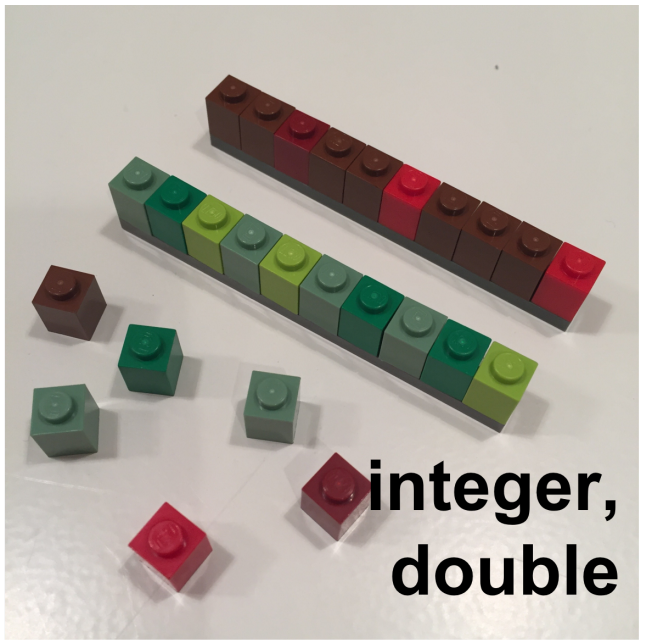
Translating R Data Types to Python

R Data Type	Description	Python Equivalent
numeric	Decimal numbers	float
integer	Whole numbers	int
character	Text or strings	str
factor	Categorical data	pandas.Categorical or str
Date	Date values	datetime.date
POSIXct	Date and time	datetime.datetime
logical	Boolean (TRUE/FALSE)	bool
complex	Complex numbers	complex

Data Structures: Atomic Vector (1D)





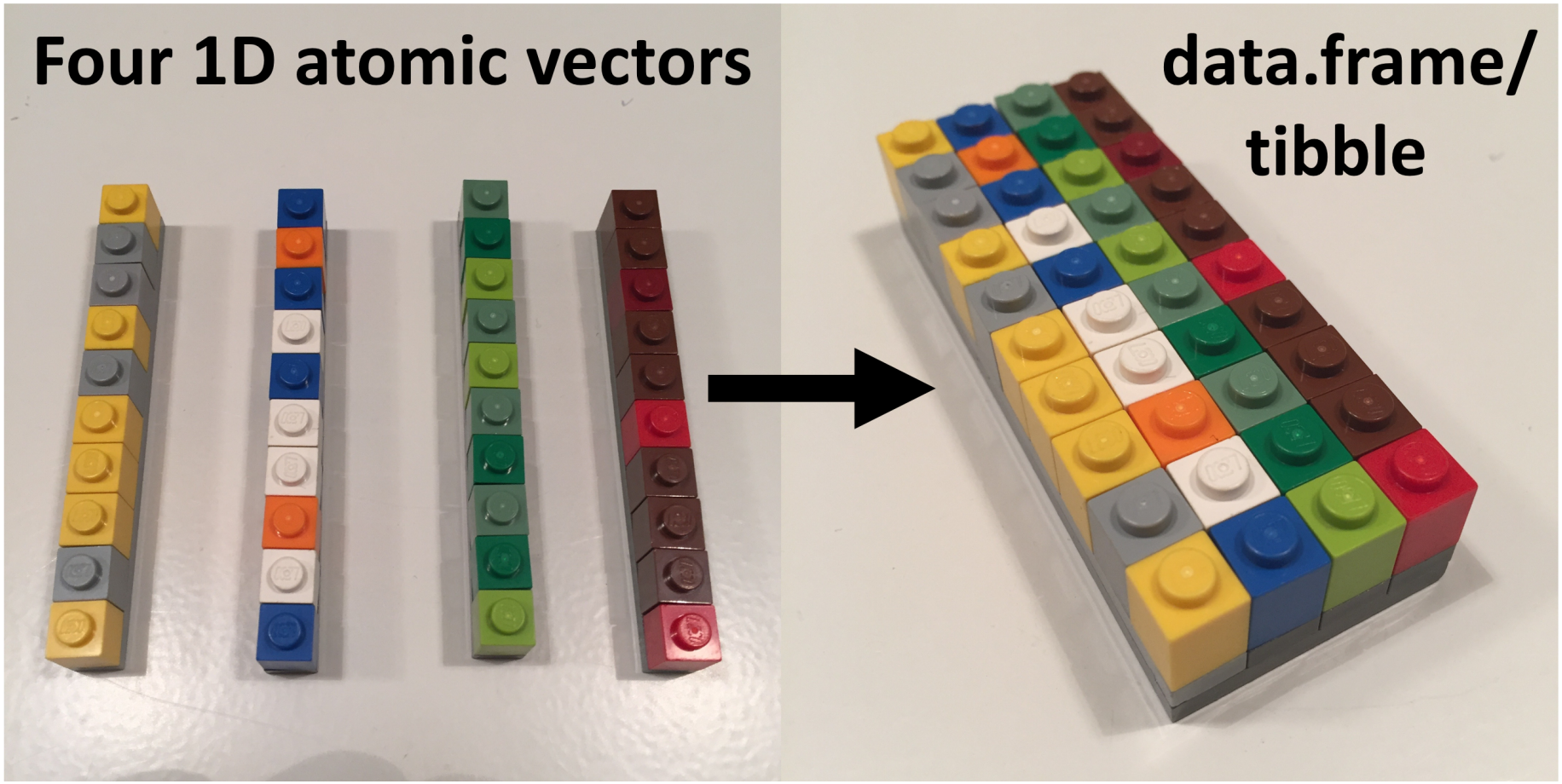


Keeping the visual representation of different types of atomic vectors in your head!!

```
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]

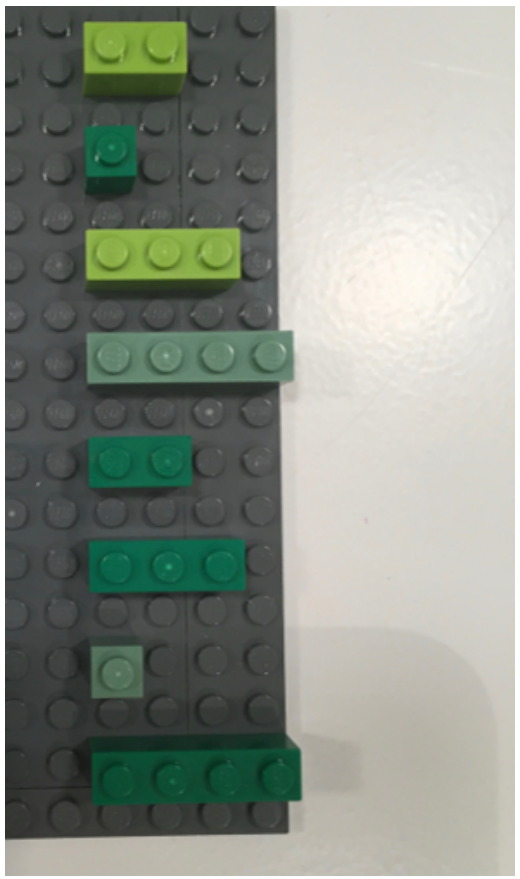
```
library(tibble)

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

```
## # A tibble: 5 × 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
```

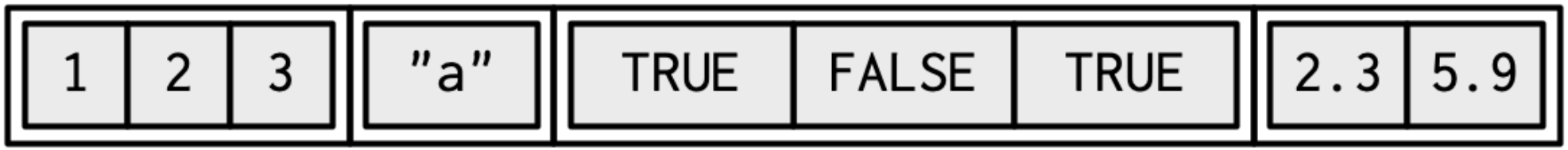
Data Structures: Lists [1]

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 `[]`

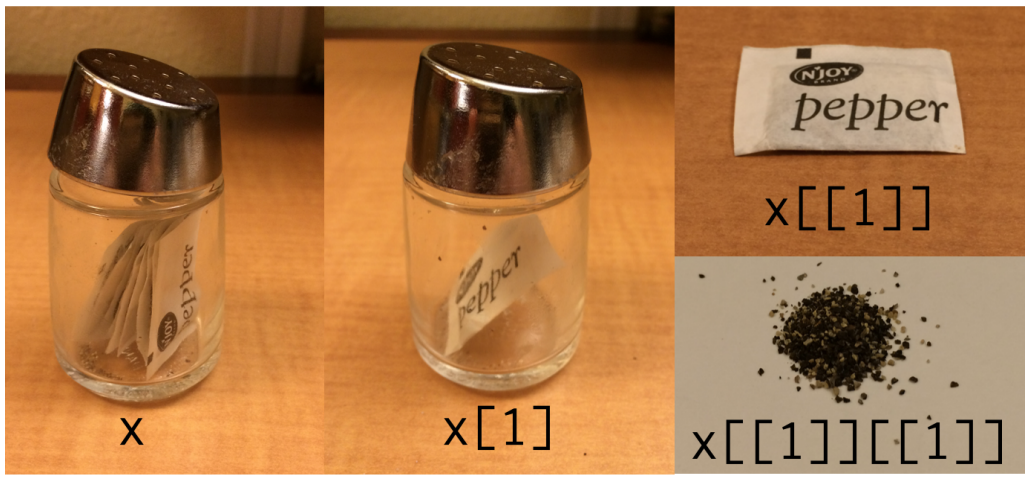
```
lst[1]
```

```
## [[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.**

```
x_mat = matrix( sample(1:10, size = 4), n
str(x_mat) # its structure?
```

```
##  int [1:2, 1:2] 9 6 2 7
```

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

```
##      [,1] [,2]
## [1,]    9    2
## [2,]    6    7
## [1] "-----------------"
## [1] 2
```

Data Structures: Matrices

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

```
x_mat = matrix( sample(1:10, size = 4), n
str(x_mat) # its structure?
```

```
##  int [1:2, 1:2] 9 6 2 7
```

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

```
##      [,1] [,2]
## [1,]    9    2
## [2,]    6    7
## [1] "-----------------"
## [1] 2
```

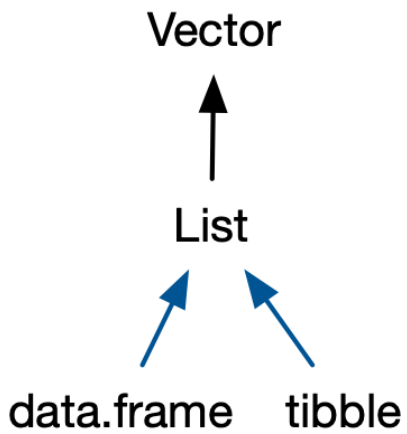
```
x_char = matrix(
  sample(letters, size = 12), nrow = 3, n
x_char
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "a"  "i"  "w"  "d"
## [2,] "u"  "j"  "g"  "r"
## [3,] "v"  "h"  "s"  "t"
```

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

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

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"
```

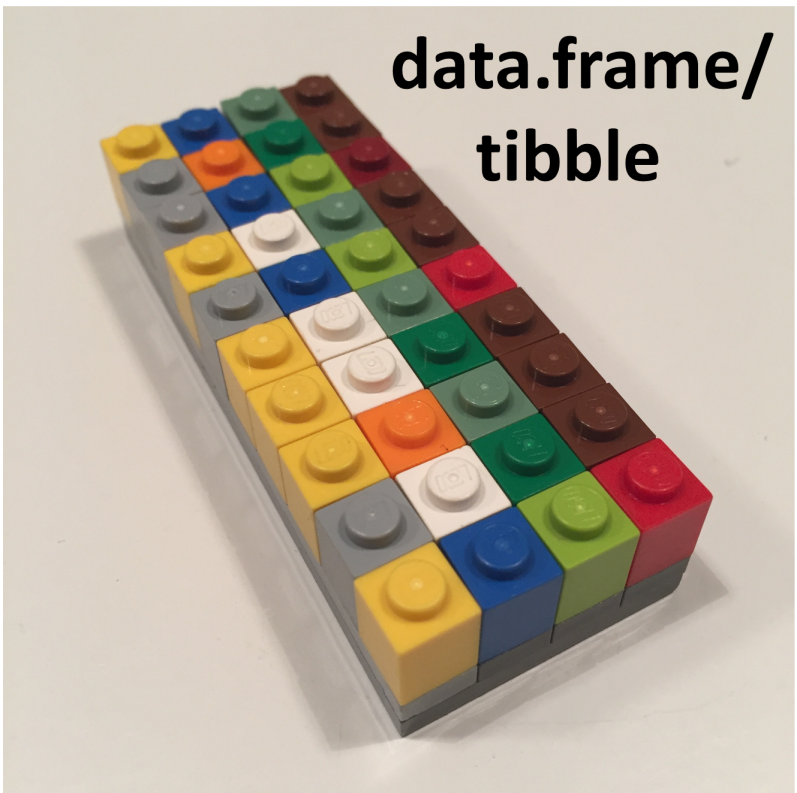
```
attributes(df1) # but also is of class data.frame
```

```
## $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.**

Data Structures: Data Frames [3]

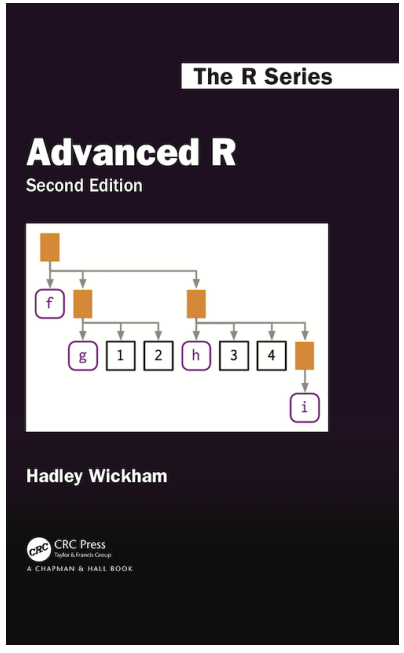
As noted in the creation of `df1`, columns in a data frame can be of different types. Hence, it is more widely used in data analysis than matrices.



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

Data Structures: So What is a Tibble?

> 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)

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)

Functions

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

```
temp_high_forecast = c(86, 84, 85, 89, 89, 84, 81)
mean(x = temp_high_forecast)
```

```
## [1] 85.42857
```

- 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] 85.42857
```

Match by **names**

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

```
## [1] 85.42857
```

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

Use Functions from Packages

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

```
## [1] 86.00000 85.00000 85.00000 86.0000
```

```
first(temp_high_forecast)
```

```
## [1] 86
```

```
last(temp_high_forecast)
```

```
## [1] 81
```

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

Write Your Own R 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] 85.42857
```

Write Your Own Python Functions

```python
# Translating the R function to Python
def my_mean(x, na_rm=False):
    """
    Calculate the mean of a list, with an option to ignore NaN values.

    Parameters:
    x (list): List of numbers
    na_rm (bool): Whether to remove NaN values before calculating the mean

    Returns:
    float: mean of the list
    """
    if na_rm:
        x = [i for i in x if i is not None]

    summation = sum(x)
    return summation / len(x)

# Test the function with a list containing None values
temp_high_forecast = [86, 84, 85, None, 89, 84, 81]
my_mean(temp_high_forecast, na_rm=True)
```

```
## 84.83333333333333
```

Generative AI: Large Language Models

Background: Artificial Intelligence

A working definition for AI

Artificial Intelligence (AI): A system that acts in a way, where people might denote as "intelligent" if another human were to do something similar.

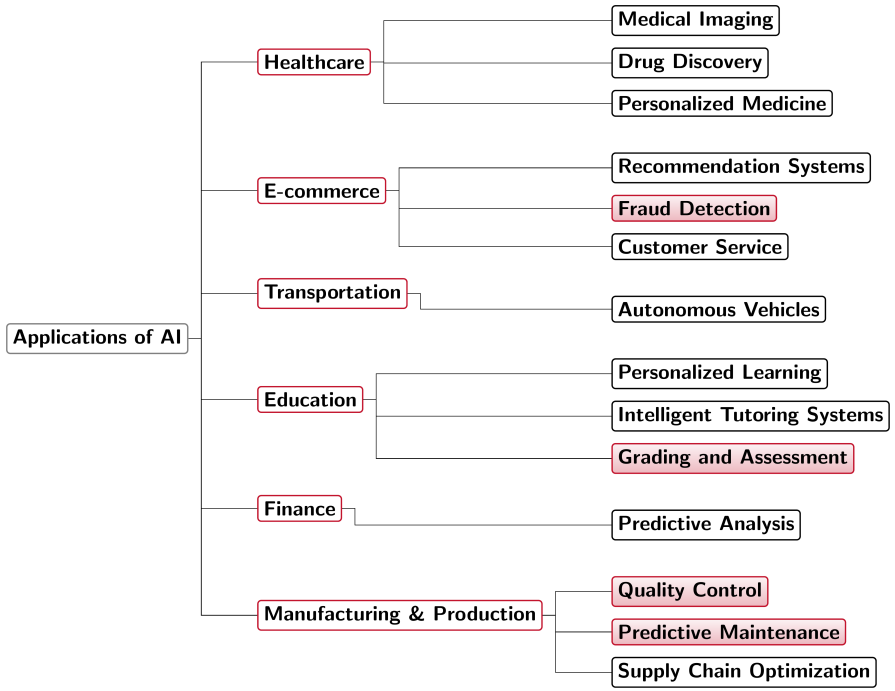
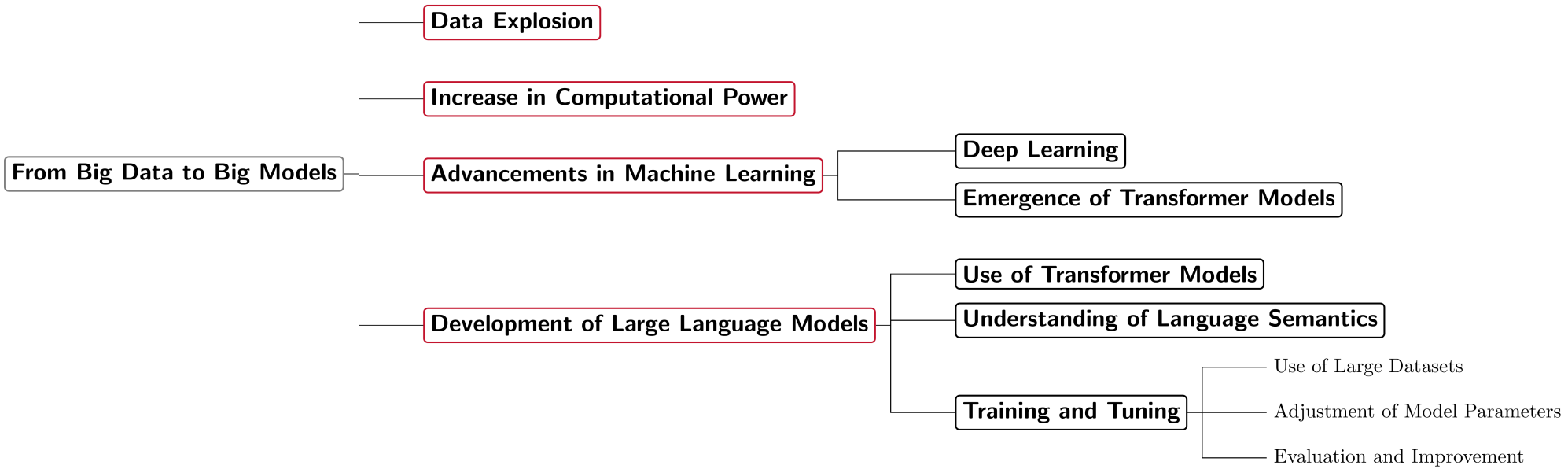


Image Source: The flowchart's content and its LaTex code were generated using ChatGPT (May 24 Version).

Background: The Road to Generative AI



Comment: You have been hearing about **big data** in SPC for over a decade now. We now have models that can digest and generate answers based on more than 45TB of text.

Background: Generative AI

Generative AI: The objective is to generate new content rather than analyze existing data.

- The generated content is based on a **stochastic behavior embedded in generative AI models such that the same input prompts results in different content**.
- State-of-the-art generative AI models can have up to **540 billion parameters** (PaLM).
- With the increase in model size, researchers have observed the **"emergent abilities"** of LLMs, which were **not explicitly encoded in the training**. Examples include:
  - Multi-step arithmetic,
  - taking college-level exams, and
  - identifying the intended meaning of a word.
- LLMs are **foundation models** (see Bommasani et al. 2021), large pre-trained AI systems that can be **repurposed with minimal effort across numerous domains and diverse tasks.**

LLMs: Natural Language Based Coding

Let us break down this prompt with ChatGPT

> I want you to help me use R to create an animated plot of the unemployment rate by state from FRED. Here are the steps that I want you to follow:
>
> - Pull the unemployment rate for all 48 states from "2003-01-01" until "2023-01-08". The symbol will be the two letter state abbreviation + UR (e.g., OHUR).
> - Add a column that contains the state name.
> - Use a choropleth map, where the unemployment rates are plotting using a sequential color scheme that is colorblind friendly (use RColorBrewer).
> - Create the animation, where each frame corresponds to a time interval. The animation should have 12 fps and show the date as part of the title.
> - Save the animation as a GIF.

Recap

Summary of Main Points

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

- 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.

Review and Clarification

- **Class Notes**: Take some time to revisit your class notes for key insights and concepts.
- **Zoom Recording**: The recording of today's class will be made available on Canvas approximately 3-4 hours after the session ends.
- **Questions**: Please don't hesitate to ask for clarification on any topics discussed in class. It's crucial not to let questions accumulate.

📖 Required Readings 📖

🤖 LLM: Prep

- AI and the Future of Work in Statistical Quality Control: ChatSQC.
  - Read the **abstract**, **Sections 1, 4, and 5**; feel free to skim sections 2-3.
  - Please feel free to test the app at: https://chatsqc.fsb.miamioh.edu/.

🎯 Assignment 🎯

- Complete Assignment 02 on Canvas to reinforce your understanding and application of the topics covered today as well as the assigned readings.