MAD – Data Analysis & Biostatistics in R

Introduction - Basics

James R. Hunter, Ph.D.

DIPA, EPM, UNIFESP

4 de setembro de 2020

Section 1

Data Types in R

Atomic Data Types

- `<int>` **integer**
  - Whole numbers (no decimal part)
  - Designated in R with a following "L" (`26L`)
- `<dbl>` **double**
  - "Real" numbers
  - Have a decimal part
  - Also referred to in R as "numeric"
- `<chr>` **character**
  - Strings
  - Surrounded by single or double quotation marks (`''` or `""`)
- `<lgl>` **logical**
  - Only take values `TRUE` or `FALSE`

Logical Data

- Can be interpreted as numbers
  - `FALSE` $= 0$
  - `TRUE` $= 1$
- Can be assigned as is
  - `x <- TRUE`
- Can be result of *logical* calculation
  - Are two numbers equal, not equal, ...

Logical Calculations

- Logical Operators
  - == left side equal to right side
  - != left side **not** equal to right side
  - >= left side greater than or equal to right side
  - < left side less than right side
- Examples

```
x <- 6 # assign a value to x
x == 6 # test if x is equal to 6
```

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

```
2 > 4
```

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

What Kind of Variable Is It?

- `typeof()` - internal type of an object
- `class()` - general type of classification
  - Most useful for composite (ie, non-atomic) types

Examples of Atomic Types

```
int_var <- 25L
typeof(int_var)
```

```
## [1] "integer"
```

```
num_var <- 25.879
typeof(num_var)
```

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

```
log_var <- TRUE
typeof(log_var)
```

```
## [1] "logical"
```

```
char_var <- "abcd"
typeof(char_var)
```

```
## [1] "character"
```

Hierarchy of Classes and Coercion

- R understands that you can't mix operations involving different classes
  - Cannot multiply a string by a number
  - R returns an error

```
> char_var * 2
Error in char_var * 2 : non-numeric argument to binary operator
```

Hierarchy

- From most restricted to most general
  - **Logical**
    - ⋆ Only take on 2 values
    - ⋆ Also *integers* (with values 1 and 0) and can be calculated
  - **Integer**
    - ⋆ Can be calculated as if they were *numeric*
    - ⋆ Stored more compactly in memory than *numeric*
  - **Numeric**
    - ⋆ Any number can be stored as *numeric*
  - **Character**
    - ⋆ Another other class can be stored as *character*
    - ⋆ Not able to perform calculations on *character*

Coercion

- Relates to primarily to *vectors*
- Vectors must be all of only one data type
- R will coerce the values to be the most general type of the types you give it

Examples of Coercion

```
(vect1 <- c(7L, 27890L))
```

```
## [1]     7 27890
```

```
typeof(vect1)
```

```
## [1] "integer"
```

- 2 integers, result type = *integer*

Examples 2

```
(vect2 <- c(7L, 27.333))
```

```
## [1]  7.000 27.333
```

```
typeof(vect2)
```

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

- Integer and numeric values, result type = *double* (numeric)

Examples 3

```
(vect3 <- c(27.333, "cat"))
```

```
## [1] "27.333" "cat"
```

```
typeof(vect3)
```

```
## [1] "character"
```

- Double and character values, result type = *character*

Examples 4

```
(vect4 <- c(TRUE, 7L, 27.333, "cat"))
```

```
## [1] "TRUE"   "7"      "27.333" "cat"
```

```
typeof(vect4)
```

```
## [1] "character"
```

- Combination of all 4 atomic types, result type = *character*

Examples 5

```
(vect5 <- c(7L, TRUE))
```

```
## [1] 7 1
```

```
typeof(vect5)
```

```
## [1] "integer"
```

- Integer and Logical values, result type = *integer*
- Summing of these values possible

```
sum(vect5)
```

```
## [1] 8
```

Coercion by Command

- You can force the coercion of a variable to a different type
  - If it is an acceptable type for the value
- `as.xxx()` commands
  - `as.logical()`
  - `as.integer()`
  - `as.numeric()`
  - `as.character()`

Examples of Coercion

```
as.character(25.3)
```

```
## [1] "25.3"
```

```
as.integer(25.3) # will lose decimal part
```

```
## [1] 25
```

```
as.numeric("25.3") # character form of a number will be transformed
```

```
## [1] 25.3
```

```
as.numeric("cat") # Not permitted; results in NA
```

```
## [1] NA
```

Section 2

More Data Classes

Factors

- Compact way of storing categorical information
- Primary manner of storing categorical variables
  - Unless they have 000's of different states
- Converts strings, numbers and even logical variables into factor
- Two parts to a factor
  - Internal list of alternative possibilities (*levels*)
  - Integers assigning each value to a level

Factor Example - 1

- Variable for *gender* in a study
- 4 possible states:
  - "male", "female", "decline to state", "other"
- gender is currently a *character* variable

```
gender <- c("female", "decline to state", "other", "male", "female", "male")
gender
```

```
## [1] "female"           "decline to state" "other"            "male"
## [5] "female"           "male"
```

```
typeof(gender)
```

```
## [1] "character"
```

Factor Example – 2

- Convert gender to a factor with `factor()`

```
genderf <- factor(gender)
genderf
```

```
## [1] female           decline to state other           male
## [5] female           male
## Levels: decline to state female male other
```

```
typeof(genderf)
```

```
## [1] "integer"
```

```
class(genderf)
```

```
## [1] "factor"
```

```
str(genderf)
```

```
##  Factor w/ 4 levels "decline to state",..: 2 1 4 3 2 3
```

```
levels(genderf)
```

```
## [1] "decline to state" "female"           "male"             "other"
```

Date and Time Representations

- Multiple data classes that deal with dates and time
- Focus on *Date* class
- Multiple packages that facilitate working with dates
- In databases, dates normally shown as strings ("04-09-2020")

Convert Date String to Date Class

- `as.Date()` function
- If date not in ISO date format ("YYYY-mm-dd")
  - Need to tell `as.Date()` what the format is
    - "%d" = day
    - "%m" = month
    - "%y" or "%Y" = year (2 digit or 4 digit)
    - Whole string enclosed in quotes ("")

Convert Date Example

```
d <- "28/05/2020"
d_date <- as.Date(d, format = "%d/%m/%Y")
d_date
```

```
## [1] "2020-05-28"
```

```
class(d_date)
```

```
## [1] "Date"
```

Date Math

- Can calculate periods of time using Date format
- Using normal arithmetic operations
- Note: `Sys.Date()` function returns current date
- Date math useful for calculating ages, time since infection

```
today <- Sys.Date() # function that returns current date
today
```

```
## [1] "2020-09-03"
```

```
today - d_date
```

```
## Time difference of 98 days
```

Section 3

Working with Vectors

Indexing Vectors

- Access any element of vector with number surrounded by "[]" (square brackets)
- Vector `x` composed of 10 real numbers.
  - First number: `x[1]`
  - Sixth number: `x[6]`

```
x <- c(177.89, 194.47, 32.24, 99.56, 205.34,
       -0.95, 171.96, 112.65, 32.93, 60.53)

# 1st number
x[1]
```

```
## [1] 177.89
```

```
# 6th number
x[6]
```

```
## [1] -0.95
```

There´s More ...

- To access the 2nd and 5th elements of `x`
  - Use the `c()` function
- To access the 2nd *through* the 5th elements
  - Use the `:` operator

```
## [1] 194.47 205.34
```

```
## [1] 194.47  32.24  99.56 205.34
```

head() and tail()

- Access first n values of vector with `head()`
  - n as argument within parentheses (default = 6)
- Access last n values of vector with `tail()`

```
head(x)
```

```
## [1] 177.89 194.47  32.24  99.56 205.34  -0.95
```

```
tail(x)
```

```
## [1] 205.34  -0.95 171.96 112.65  32.93  60.53
```

```
head(x, n = 3)
```

```
## [1] 177.89 194.47  32.24
```

Section 4

Data Classes for Databases

What Is a Database in R?

- Collection of variables
  - Each of which has a number of cases
- Variable
  - Identifier
  - Categorical classifier
  - Numerical or logical value
- Variable is a Vector
  - All values must be the same type
  - All variables in a database (except *list*) must be the same length
- 3 Classes of Databases
  - Lists
  - Dataframes
  - Tibbles

Lists

- Most general type
- Any data type
- R will preserve it without coercion
- Not used too much in normal data analysis

```
vect4 <- list(TRUE, 7L, 27.333, "cat")
typeof(vect4)
```

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

```
vect4[[2]]
```

```
## [1] 7
```

```
typeof(vect4[[2]])
```

```
## [1] "integer"
```

Data Frames

- Most common data type for databases
- A list with additional structure (deep in R's guts)
- Variables - Columns
- Cases - Rows
- All variables must be same length
- Variables have names
- Cases *can* have names (not required)
- View structure with `str()`

```r
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```r
head(mtcars, n = 3)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```r
class(mtcars)
```

```
## [1] "data.frame"
```

```r
typeof(mtcars)
```

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

Build a Data Frame

- Use the function `data.frame()`
  - List the variables and values you want them to take on.
  - Separate the variables by commas
  - R assigns data types to the variables we create

```r
df <- data.frame(name = c("Jim", "Fernanda", "Ana"),
                 gender = c("male", "female", "female"),
                 score = c(89, 91, 93),
                 passed = c(TRUE, TRUE, TRUE))
str(df)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ name  : chr  "Jim" "Fernanda" "Ana"
##  $ gender: chr  "male" "female" "female"
##  $ score : num  89 91 93
##  $ passed: logi  TRUE TRUE TRUE
```

$ Notation

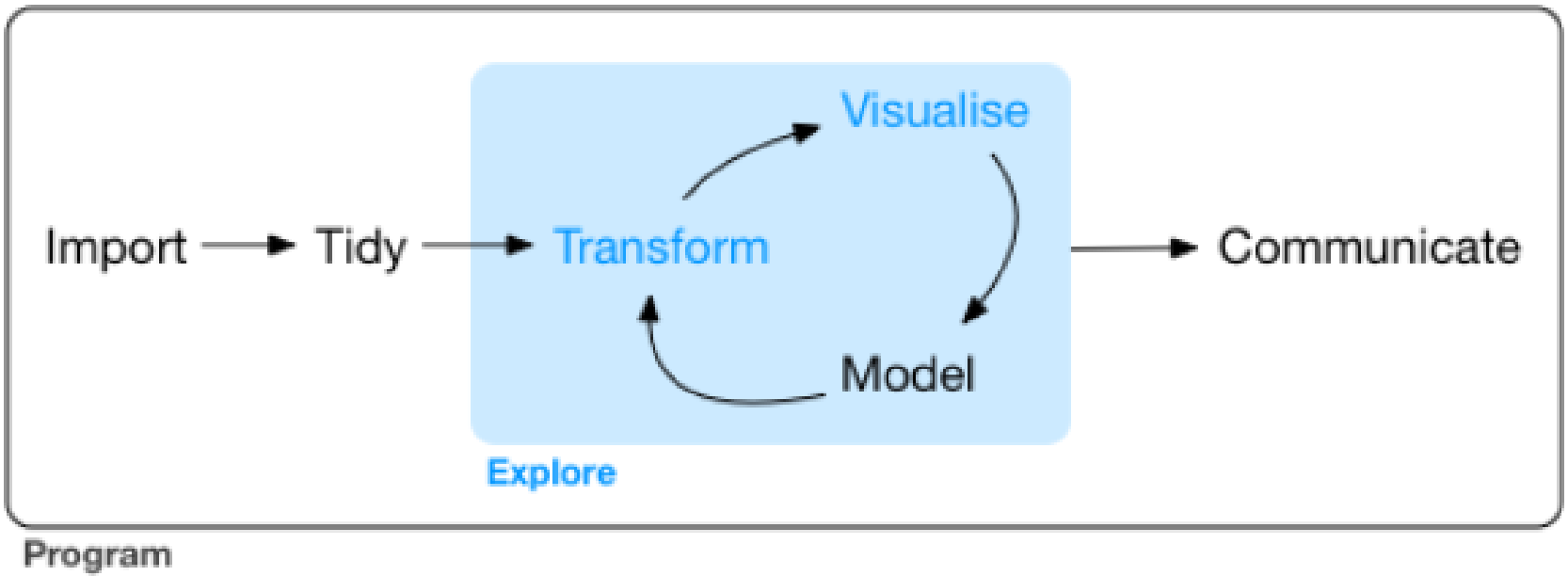
- What is the $ doing in the str() function?
  - `$ name  : chr  "Jim" "Fernanda" "Ana"`
- $ separates the data frame name from the variable name
- `df$name` will produce all three names
  - Jim, Fernanda, Ana
- Bracket notation at the end can select among the names
  - `df$name[2]` produces Fernanda

Tibbles

- Advanced form of data frame
- Part of the **Tidyverse**
- Use the same rules as data frames
- Somewhat different internal structure
- Faster processing (most of the time)

Section 5

Data Analysis Workflow - The Big Picture


Visualise
Import
Tidy
Transform
Communicate
Model
Explore
Program

Import Data

- First step in any data analysis project
- Sources of data:
  - Excel files (.csv or .xlsx)
  - Other text files (.txt or .csv)
  - Fasta files
  - Files from other machines
- Very little data directly prepared in R

Tidy Data

- Ensure that data appears in a useful form for analysis
- Most data sets are collected and stored haphazardly
  - Many with missing data points
  - Impossible values
    - The “135” year old man
- Need to catch these errors
  - Correct what is possible
  - Clear strategy for dealing with the rest

“Tidy Data”

- Set of principles about organization of data for analysis
- “Tidyverse”
  - Set of “opiniated” packages that promote and use principles of “tidy” formatting
- Hallmark of “tidy data” is consistency


stringr
forcats
tidyverse
www.rstudio.com
dplyr
tidyr
lubridate
FIGGLE
readr
purrr
Horst

Hadley Wickham's Definition of Tidy Data

Like families, tidy datasets are all alike, but every messy dataset is messy in its own way. . . . A dataset is a collection of values, usually either numbers (if quantitative) or strings (if qualitative). Values are organized in two ways. Every value belongs to a variable and an observation. A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation contains all values measured on the same unit (like a person, or a day, or a race) across attributes. . . . Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types.[1]

[1] Wickham, Hadley. "Tidy Data". Journal of Statistical Software 59, 10 (2014). https://doi.org/10.18637/jss.v059.i10.

The Three Rules of "Tidy" Data

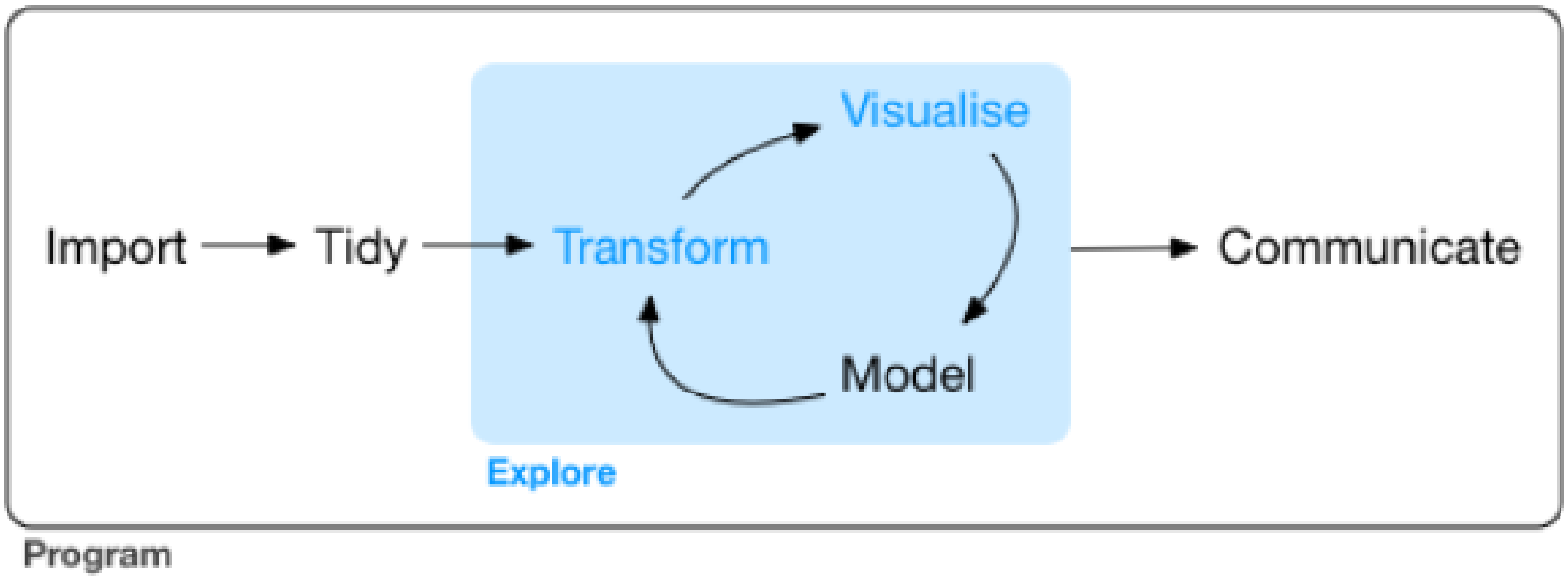
- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell[2]

variables

observations

values

[2]Wickham and Grolemund, Ch. 12.1.


Visualise
Import
Tidy
Transform
Communicate
Model
Explore
Program

Explore Data

- This is the phase you always thought was **all** of data analysis.
- Build models
- Evaluate models
- Test hypotheses
- Make graphs
  - Exploratory (for you)
  - Presentation (for public)

Communicate Results

- Write reports, presentations, etc.
- R Markdown invaluable aid in all this
  - All slides and chapters in this course prepared in R Markdown

Real Division of Labor

- Import and Tidy Data - **60 - 70 %**
- Explore Data - 20 - 30 %
- Communicate Results - 10 - 20 % ## Real Division of Labor

Next Week

- Put Import, Tidy and Explore into Practice