Getting the data in R

Julien Martin
University of Ottawa

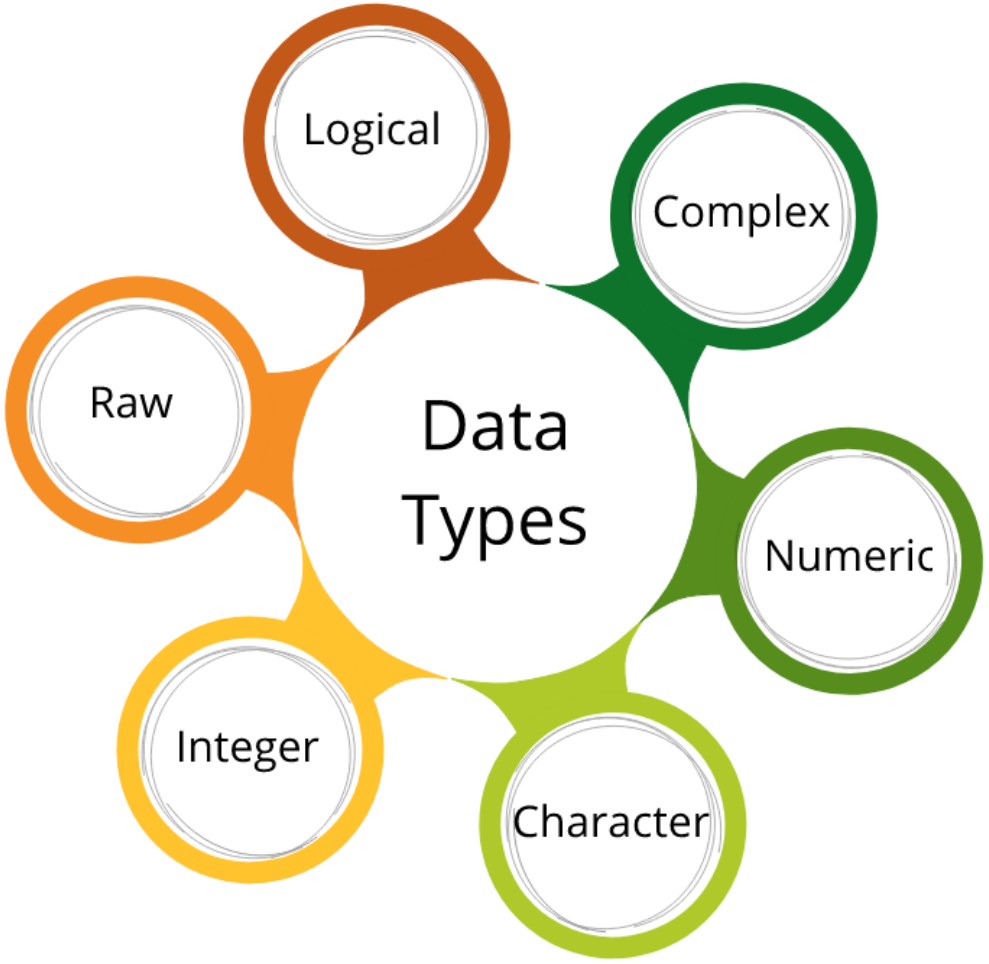
Learning outcomes

- recognise different types of data in R ✔️
- understand some different data structures ✔️
- learn how to import data into R ✔️
- learn how to manipulate data in R ✔️
- learn how to export data from R ✔️

types of data in R

six types of data in R

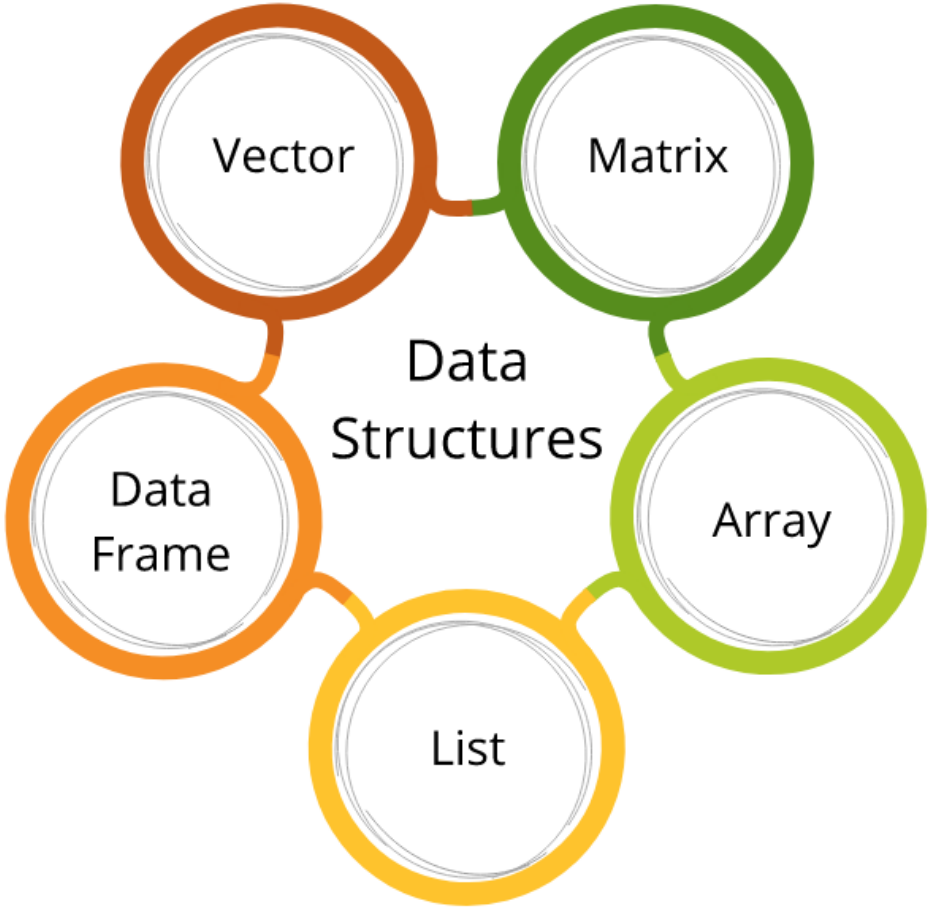
> **numeric** - 1.618, 3.14, 2.718
>
> **integers** - 1, 2, 3, 42, 101
>
> **logical** - TRUE or FALSE
>
> **character** - "BI5009", "Blue"
>
> **complex**
>
> **raw**



data structures

five data structures

> **vector**
>
> **matrix**
>
> **array**
>
> **data frame**
>
> **list**



vectors

- one dimensional collection elements
- can contain all data types
- all elements must be of the same type

```
> num <- 42
> numbers <- c(2, 3, 4, 5, 6)
> char <- c("red", "green")
> log <- c(TRUE, TRUE, FALSE)
> my_na <- c(NA, NA, NA, NA)
> mix <- c(1, 2, 3, NA, 5)
```

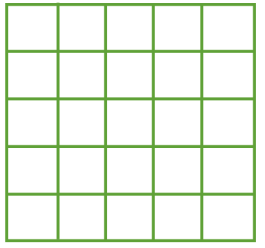
scalar vector

matrices and arrays

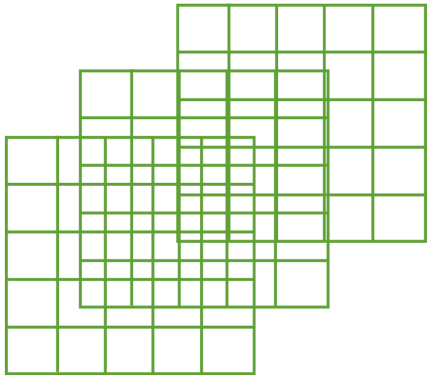
- a vector with extra dimensions
- again, objects must be of the same type
- arrays are multidimensional matrices

```
> mat.1 <- matrix(1:12, nrow=4)
> mat.1
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
> array.1 <- array(1:16, dim=c(2,4,2))
```



matrix



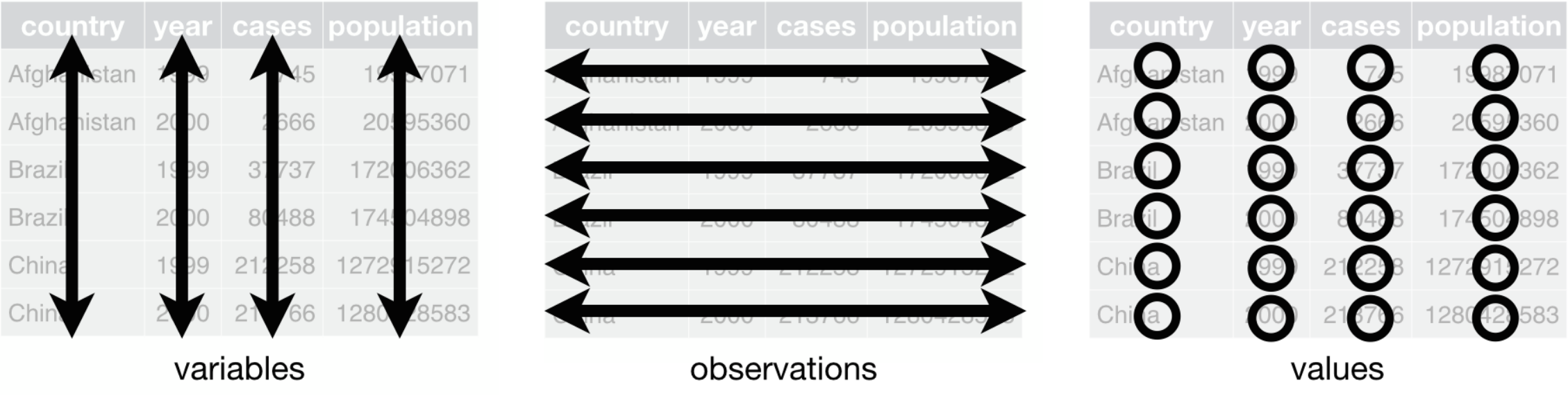
array

data frames

- most commonly used data structure for statistical data analysis
- powerful 2-dimensional vector holding structure
- dataframes can hold vectors of any of the basic classes of data

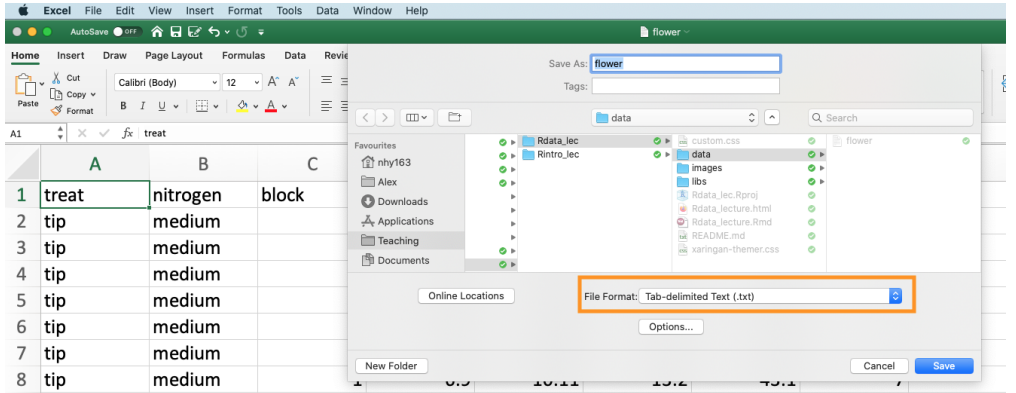
```
   treat nitrogen block height weight leafarea shootarea flowers
1    tip   medium     1    7.5   7.62     11.7      31.9       1
2    tip   medium     1   10.7  12.14     14.1      46.0      10
3    tip   medium     1   11.2  12.76      7.1      66.7      10
4    tip   medium     1   10.4   8.78     11.9      20.3       1
5    tip   medium     1   10.4  13.58     14.5      26.9       4
6    tip   medium     1    9.8  10.08     12.2      72.7       9
7    tip   medium     1    6.9  10.11     13.2      43.1       7
8    tip   medium     1    9.4  10.28     14.0      28.5       6
9    tip   medium     2   10.4  10.48     10.5      57.8       5
10   tip   medium     2   12.3  13.48     16.1      36.9       8
```

tidy data


country
year
cases
population
variables
observations
values

importing data

- simplest method is to use spreadsheet and then import data into R
- use either MS Excel or LibreOffice calc
- File --> Save as ... menu
- save as a comma separated file (*.csv)
- missing data represented with NA
- no spaces in names (use underscore if needed)
- no uppercase
- keep variable names short & informative



	A	B	C					
1	treat	nitrogen	block					
2	tip	medium						
3	tip	medium						
4	tip	medium						
5	tip	medium						
6	tip	medium						
7	tip	medium						
8	tip	medium						
9	tip	medium	1	9.4	10.28	14	28.5	6
10	tip	medium	2	10.4	10.48	10.5	57.8	5
11	tip	medium	2	12.3	13.48	16.1	36.9	8
12	tip	medium	2	10.4	13.18	11.1	56.8	12
13	tip	medium	2	11	11.56	12.6	31.3	6
14	tip	medium	2	7.1	8.16	29.6	9.7	2
15	tip	medium	2	6	11.22	13	16.4	3
16	tip	medium	2	9	10.2	10.8	90.1	6
17	tip	medium	2	4.5	12.55	13.4	14.4	6
18	tip	high	1	12.6	18.66	18.6	54	9

importing data

- the `read.table()` function is the workhorse with format specified function to make coding faster

```
petunia <- read.table('data/flowers.csv', header = TRUE, sep = ',')
```

columns separated with a ***comma***

importing data

- sometimes columns are separated by tabs

```
petunia <- read.table('data/flowers.csv', header = TRUE, sep = '\t')`
```

- you can use format specific functions

```
petunia <- read.csv('flowers.csv') # if comma-separated

petunia <- read.delim('flowers.txt') # if tab-separated
```

- functions in the `foreign` package allows you to import files of other formats (i.e. from SAS, SPSS, Minitab etc)

- use the `readxl` or `readODS` package to directly import MS Excel or LO calc spreadsheets directly

importing data

- to view the contents of a data frame, type it's name
- rarely a good idea as just fills up your console
- use the `head()` and `tail()` to see first/last 6 lines

```
[1] flowers
  treat nitrogen block height weight leafarea shootarea flowers
1   tip   medium     1    7.5   7.62     11.7      31.9       1
2   tip   medium     1   10.7  12.14     14.1      46.0      10
3   tip   medium     1   11.2  12.76      7.1      66.7      10
4   tip   medium     1   10.4   8.78     11.9      20.3       1
5   tip   medium     1   10.4  13.58     14.5      26.9       4
6   tip   medium     1    9.8  10.08     12.2      72.7       9
```

data wrangling

- `str()` function: data frame dimensions, list of variables, type of variables

```
> str(flowers)
'data.frame':    96 obs. of  8 variables:
 $ treat    : chr  "tip" "tip" "tip" "tip" ...
 $ nitrogen : chr  " medium" " medium" " medium" " medium" ...
 $ block    : int  1 1 1 1 1 1 1 1 2 2 ...
 $ height   : num  7.5 10.7 11.2 10.4 10.4 9.8 6.9 9.4 10.4 12.3 ...
 $ weight   : num  7.62 12.14 12.76 8.78 13.58 ...
 $ leafarea : num  11.7 14.1 7.1 11.9 14.5 12.2 13.2 14 10.5 16.1 ...
 $ shootarea: num  31.9 46 66.7 20.3 26.9 72.7 43.1 28.5 57.8 36.9 ...
 $ flowers  : int  1 10 10 1 4 9 7 6 5 8 ...
```

- `names()` function: vector of variable names

```
> names(flowers)
[1] "treat"     "nitrogen"  "block"     "height"    "weight"    "leafarea"
[7] "shootarea" "flowers"
```

data wrangling

- access variables in your data frame using the `$` notation

```
> flowers$height
 [1]  7.5 10.7 11.2 10.4 10.4  9.8  6.9  9.4 10.4 12.3 10.4 11.0  7.1  6.0  9.0
[16]  4.5 12.6 10.0 10.0  8.5 14.1 10.1  8.5  6.5 11.5  7.7  6.4  8.8  9.2  6.2
[31]  6.3 17.2  8.0  8.0  6.4  7.6  9.7 12.3  9.1  8.9  7.4  3.1  7.9  8.8  8.5
[46]  5.6 11.5  5.8  5.6  5.3  7.5  4.1  3.5  8.5  4.9  2.5  5.4  3.9  5.8  4.5
[61]  8.0  1.8  2.2  3.9  8.5  8.5  6.4  1.2  2.6 10.9  7.2  2.1  4.7  5.0  6.5
[76]  2.6  6.0  9.3  4.6  5.2  3.9  2.3  5.2  2.2  4.5  1.8  3.0  3.7  2.4  5.7
[91]  3.7  3.2  3.9  3.3  5.5  4.4
```

- you can extract elements in the data frame using the `[rowIndex, columnIndex]` method
- `Index` can either be a positional index or a logical index

positional index

- provide the row and column position of the data you wish to extract
- `index` can either be a positional index or a logical index

```
> flowers[1, 4]      # extract value of first row and 4th column
[1] 7.5
```

- extract multiple elements by supplying vectors for `rowIndex` and `columnIndex`

```
> flowers[1:3, 1:4]      # extract rows 1 to 3 and columns 1 to 4
  treat nitrogen block height
1   tip   medium     1    7.5
2   tip   medium     1   10.7
3   tip   medium     1   11.2
```

positional index

- another example

```
> flowers[c(3,8,20), c(1, 4, 5, 6)]     # rows 3, 8 and 20 and columns 1, 4, 5 and 6
   treat height weight leafarea
3    tip   11.2  12.76      7.1
8    tip    9.4  10.28     14.0
20   tip    8.5  14.33     13.2
```

- can assign these extracted values to another object if you want
- new object inherits `data.frame` class

```
> flowers_red <- flowers[c(3,8,20), c(1, 4, 5, 6)]
> flowers_red
   treat height weight leafarea
3    tip   11.2  12.76      7.1
8    tip    9.4  10.28     14.0
20   tip    8.5  14.33     13.2
```

positional index

- we can use a short cut if we want all rows or all columns extracted
- omitting the column index is shorthand for 'all columns'

```
> flowers[1:3, ]
  treat nitrogen block height weight leafarea shootarea flowers
1   tip   medium     1    7.5   7.62     11.7      31.9       1
2   tip   medium     1   10.7  12.14     14.1      46.0      10
3   tip   medium     1   11.2  12.76      7.1      66.7      10
```

- omitting the row index is shorthand for 'all rows'

```
> flowers[, 1:3]
```

positional index

- an alternative method to select columns is to name the columns directly for `columnIndex`

```
> flowers[1:10, c('treat', 'nitrogen', 'leafarea', 'shootarea')]
   treat nitrogen leafarea shootarea
1    tip   medium     11.7      31.9
2    tip   medium     14.1      46.0
3    tip   medium      7.1      66.7
4    tip   medium     11.9      20.3
5    tip   medium     14.5      26.9
6    tip   medium     12.2      72.7
7    tip   medium     13.2      43.1
8    tip   medium     14.0      28.5
9    tip   medium     10.5      57.8
10   tip   medium     16.1      36.9
```

logical index

- we can also extract rows based on a logical test
- example, let's extract all rows where the `height` variable is greater than 12

```
> flowers[flowers$height > 12,]
   treat nitrogen block height weight leafarea shootarea flowers
10   tip   medium     2   12.3  13.48     16.1      36.9       8
17   tip     high     1   12.6  18.66     18.6      54.0       9
21   tip     high     1   14.1  19.12     13.1     113.2      13
32   tip     high     2   17.2  19.20     10.9      89.9      14
38   tip      low     1   12.3  11.27     13.7      28.7       5
```

- or where `leafarea` is equal to 8.7

```
> flowers[flowers$leafarea == 8.7,]
   treat nitrogen block height weight leafarea shootarea flowers
35   tip      low     1    6.4   5.97      8.7       7.3       2
45   tip      low     2    8.5   7.16      8.7      29.9       4
```

logical index

- we can combine logical tests using the `&` symbol (AND) or the `|` symbol (OR)
- example, extract all rows where `height` is > 10.5 and `nitrogen` is equal to `"medium"`

```
> flowers[flowers$height > 10.5 & flowers$nitrogen == 'medium',]
[1] treat     nitrogen  block     height    weight    leafarea  shootarea
[8] flowers
<0 rows> (or 0-length row.names)
```

- or `height` is greater than 12.3 OR less than 1.8

```
> flowers[flowers$height > 12.3 | flowers$height < 1.8,]
   treat nitrogen block height weight leafarea shootarea flowers
17   tip     high     1   12.6  18.66     18.6      54.0       9
21   tip     high     1   14.1  19.12     13.1     113.2      13
32   tip     high     2   17.2  19.20     10.9      89.9      14
68 notip     high     1    1.2  18.24     16.6     148.1       7
```

exporting data frames

- the `write.csv()` function exports data frames to an external file

```
write.csv(flowers, 'flowers2.csv', row.names = FALSE)
```

- saves `flowers` data frame to a file named 'flowers.csv'
- `row.names = FALSE` argument supresses the row names in the file

other options

- there are many other options for importing and exporting data in R
- the `fread()` and `fwrite()` functions in the `read.table` package are blazingly fast
- the `read_csv()` and `write_csv()` (and other related) functions from the `readr` package for tidyverse alternatives
- if you have a lot of data (I mean alot!) then take a look at the `ff` and `bigmemory` packages

Thanks!

I created these slides with xaringan and R Markdown using the rutgers css that I slightly modified.

Credit: I borrowed slides from Alex Douglas.