

# Basic Data Structures in R

HES 505 Fall 2022: Session 2

Matt Williamson

# Checking in

1. What are some advantages and disadvantages of using R for spatial analysis
2. What can I clarify about the course?
3. How do you feel about git and github classroom? How can I make that easier for you?

# Today's Plan

- Understanding data types and their role in R
- Reading, subsetting, and manipulating data
- Getting help
- First assignment is live!

# Data types and structures

# Data types

- The basic schema that **R** uses to store data.
- Creates expectations for allowable values
- Sets the “rules” for how your data can be manipulated
- Affects storage and combination with other data types
- Four most common: **Logical, Numeric, Integer, Character**

# Logical Data

- Data take on the value of either **TRUE** or **FALSE**.
- Special type of logical called **NA** to represent missing values
- Can be coerced to integers when numeric data is required (**TRUE** = 1; **FALSE** = 0)

# Logical Data (cont'd)

- Can be the outcome of logical test

```
1 x <- runif(10,-10, 10) #generate 10 random numbers between -10 and 10
2 (y <- x > 5) #test whether the values are greater than 5 and assign to obje
[1] TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
1 typeof(y) #how is R storing the object?
[1] "logical"
1 mean(y) #gives the proportion of y that is greater than 5
[1] 0.3
1 x[c(3,6,8)] <- NA #set the 3rd, 6th, and 8th value to NA
2 is.na(x) #check which values are NA
[1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
```

# Numeric Data

- All of the elements of an object (or variable) are numbers that *could* have decimals
- R can store this as either *double* (at least 2 decimal points) or *integer*

```
1 x <- runif(10,-10, 10) #generate 10 random numbers between -10 and 10  
2 typeof(x) #how is R storing the object?
```

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

```
1 class(x) #describes how R will treat the object
```

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

# Integer Data

- Integer data is a special case of numeric data with no decimals

```
1 mode(x) <- "integer"  
2 x
```

```
[1] 7 4 -1 3 -6 9 3 0 6 -1
```

```
1 class(x)
```

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

```
1 typeof(x)
```

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

```
1 z <- sample.int(100, size=10) #sample 10 integers between 1 and 100  
2 typeof(z)
```

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

```
1 class(z)
```

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

# Character Data

- Represent *string* values
- Strings tend to be a word or multiple words
- Can be used with logical tests

```
1 char <- c("Sarah", "Tracy", "Jon") #use c() to combine multiple entries  
2 typeof(char)
```

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

```
1 char == "Jon"
```

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

```
1 char[char=="Jon"] <- "Jeff"  
2 char
```

```
[1] "Sarah" "Tracy" "Jeff"
```

# Factors

- A special case of character data
- Data contains a limited number of possible character strings (categorical variables)
- The **levels** of a factor describe the possible values (all others coerced to **NA**)

```
1 (sex <- factor(c("female", "female", "male", "female", "male")))) #by default  
[1] female female male   female male  
Levels: female male  
  
1 (sex <- factor(sex, levels = c("male", "female")))) #changing the order of the levels  
[1] female female male   female male  
Levels: male female
```

# Coercion

- Sometimes certain functions require a particular **class** of data require conversion (or coercion)
- **mode** - implicitly; **as.xxxx** - explicitly

```
1 text <- c("test1", "test2", "test1", "test1") # create a character vector  
2 class(text)
```

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

```
1 text_factor <- as.factor(text) # transform to factor  
2 class(text_factor) # recheck the class
```

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

```
1 levels(text_factor)
```

```
[1] "test1" "test2"
```

```
1 as.numeric(text_factor)
```

```
[1] 1 2 1 1
```

# Data structures

- Lots of options for how R stores data
- Structure determines which functions work and how they behave
- `length()`, `str()`, `summary()`, `head()`, and `tail()` can help you explore
- Most of the `RSpatial` data structures build on these basic structures

# Vectors

- A 1-dimensional collection of elements with the same data type
- Combining two datatypes makes R choose

```
1 series.1 <- seq(10)
2 series.2 <- seq(from = 0.5, to = 5, by = 0.5)
3 series.abc <- letters[1:10]
4 length(series.1)
```

```
[1] 10
```

```
1 length(series.2)
```

```
[1] 10
```

```
1 class(c(series.abc, series.1)) #combine characters with numbers
```

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

# Vectors (cont'd)

- Can combine them or perform ‘vectorized’ operations

```
1 series.comb <- c(series.1, series.2)
2 length(series.comb)
```

```
[1] 20
```

```
1 series.add <- series.1 + series.2
2 length(series.add)
```

```
[1] 10
```

```
1 head(series.add)
```

```
[1] 1.5 3.0 4.5 6.0 7.5 9.0
```

- What happens if you try to add the character vector to the numeric vector?

# Matrices

- An extension of the numeric or character vectors to include 2-dimensions (rows and columns)
- Arrays extend the idea to multiple dimensions
- Elements of matrix must have the same data type

```
1 (m <- matrix(1:6, nrow = 2, ncol = 3)) #default is to fill by columns
```

```
 [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
1 dim(m)
```

```
[1] 2 3
```

```
1 (m <- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE))
```

```
 [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

# Lists

- Hold a variety of different data types and structures including more lists.
- Use a lot for functional programming (next week).

```
1 (xlist <- list(a = "Waldo", b = 1:10, data = head(mtcars)))  
  
$a  
[1] "Waldo"  
  
$b  
[1] 1 2 3 4 5 6 7 8 9 10  
  
$data  
          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  
Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44  1  0    3    1  
Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02  0  0    3    2  
Valiant      18.1   6 225 105 2.76 3.460 20.22  1  0    3    1
```

# Lists (cont'd)

- Lists store information in **slots**
- Adding **names** to a list can help with accessing data

```
1 names(xlist)
```

```
[1] "a"     "b"     "data"
```

```
1 class(xlist$data)
```

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

# Data Frames

- Resemble tabular datasets used in spreadsheet programs
- Long vs. wide data
- Special type of list where every element has the same length (but can have different types of data)

```
1 (dat <- data.frame(id = letters[1:5], x = 1:5, y = rep(date(), times=5 )))
```

```
id x                  y
1  a 1 Tue Aug 15 09:43:53 2023
2  b 2 Tue Aug 15 09:43:53 2023
3  c 3 Tue Aug 15 09:43:53 2023
4  d 4 Tue Aug 15 09:43:53 2023
5  e 5 Tue Aug 15 09:43:53 2023
```

```
1 is.list(dat)
```

```
[1] TRUE
```

```
1 class(dat)
```

# Data Frames (cont'd)

- Lots of ways to access and summarize data in data frames
- Useful for making sure your functions are working as intended

```
1 str(dat) #compact summary of the structure of a dataframe
```

```
'data.frame': 5 obs. of 3 variables:  
 $ id: chr  "a" "b" "c" "d" ...  
 $ x : int  1 2 3 4 5  
 $ y : chr  "Tue Aug 15 09:43:53 2023" "Tue Aug 15 09:43:53 2023" "Tue Aug 15  
 09:43:53 2023" "Tue Aug 15 09:43:53 2023" ...
```

```
1 summary(dat) #estimate summary statistics of data frame
```

	id	x	y
Length:5	Min. :1	Length:5	
Class :character	1st Qu.:2	Class :character	
Mode :character	Median :3	Mode :character	
	Mean :3		

# Data Frames (one more time)

- Special cases of **names** (**colnames** and **rownames**)

```
1 colnames(dat) #get the names of the variables stored in the data frame  
[1] "id" "x"   "y"  
  
1 dat$y  
[1] "Tue Aug 15 09:43:53 2023" "Tue Aug 15 09:43:53 2023"  
[3] "Tue Aug 15 09:43:53 2023" "Tue Aug 15 09:43:53 2023"  
[5] "Tue Aug 15 09:43:53 2023"
```

# Tibbles

- Similar to data frames, but allow for *lists within columns*
- Designed for use with the **tidyverse**
- Foundation of **sf** objects

```
1 library(tidyverse) #load the package necessary  
2 dat.tib <- tibble(dat)  
3 is.list(dat.tib)
```

```
[1] TRUE
```

```
1 ## [1] TRUE  
2  
3 class(dat.tib)
```

```
[1] "tbl_df"      "tbl"        "data.frame"
```

# Manipulating data in R

# A Note on the tidyverse

- A self-contained universe of packages and functions designed to work together
- Rely on “verbs” to make coding more intuitive
- Benefits and drawbacks

# Reading Data

- The first step in any data analysis
- Depends on the file type (**.csv**, **.txt**, **.shp**)
- **CHECK YOURSELF**

```
1 cars <- read.table('file/cars.txt')
2 str(cars)

'data.frame': 50 obs. of 2 variables:
$ speed: int 4 4 7 7 8 9 10 10 10 11 ...
$ dist : int 2 10 4 22 16 10 18 26 34 17 ...
```

```
1 summary(cars)
```

	speed	dist
Min. :	4.0	2.00
1st Qu.:	12.0	26.00
Median :	15.0	36.00
Mean   :	15.4	42.98
3rd Qu.:	19.0	56.00
Max.   :	25.0	120.00

# Reading Data (cont'd)

- **tidyverse** convention is to use “verb\_object”
- For reading data that means **read\_** instead of **read**.
- Different default behaviors!!

```
1 cars_tv <- read_table('file/cars.txt')
2 str(cars_tv)

spc_tbl_ [50 × 2] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
$ "speed": chr [1:50] "\"1\"" "\"2\"" "\"3\"" "\"4\"" ...
$ "dist" : num [1:50] 4 4 7 7 8 9 10 10 10 11 ...
- attr(*, "problems")= tibble [50 × 5] (S3: tbl_df/tbl/data.frame)
..$ row      : int [1:50] 1 2 3 4 5 6 7 8 9 10 ...
..$ col      : chr [1:50] NA NA NA NA ...
..$ expected: chr [1:50] "2 columns" "2 columns" "2 columns" "2 columns" ...
..$ actual   : chr [1:50] "3 columns" "3 columns" "3 columns" "3 columns" ...
..$ file     : chr [1:50] "'file/cars.txt'" "'file/cars.txt'" ...
"'file/cars.txt'" "'file/cars.txt'" ...
- attr(*, "spec")=
.. cols(
```

```
..  ` "speed" ` = col_character(),
..  ` "dist" ` = col_double()
.. )
```

# Reading Data (cont'd)

```
1 summary(cars_tv)
```

```
"speed"                  "dist"  
Length:50                Min.   : 4.0  
Class  :character        1st Qu.:12.0  
Mode   :character        Median  :15.0  
                           Mean   :15.4  
                           3rd Qu.:19.0  
                           Max.   :25.0
```

```
1 head(cars_tv)
```

```
# A tibble: 6 × 2  
`"speed"` `dist`  
<chr>      <dbl>  
1 "\"1\""  
2 "\"2\""  
3 "\"3\""  
4 "\"4\""  
5 "\"5\""  
6 "\"6\""
```

What do you notice??

# Selecting Data

- We often want to access subsets of our data
- For named objects we can use \$

```
1 speed <- cars$speed #assign the whole speed column to an object  
2 head(speed)
```

```
[1] 4 4 7 7 8 9
```

# Selecting Data (cont'd)

- More generally we can use [] (can use index and logicals)

```
1 (speed2 <- cars$speed[2]) # get the vector named speed and take the 2nd ele  
[1] 4
```

```
1 (speed3 <- cars[4,2]) #get the vector located in the 2nd column and take th  
[1] 22
```

```
1 (speed20 <- cars[cars$speed > 20,]) #return all columns where speed >20
```

	speed	dist
44	22	66
45	23	54
46	24	70
47	24	92
48	24	93
49	24	120
50	25	85

# Selecting Data (cont'd)

- For lists we use `[[]]` to access a particular slot and `[]` to access data in that slot

```
1 xlist <- list(a = "Waldo", b = 1:10, data = head(mtcars))
2 xlist[[3]][1,2] #get the 3rd slot in the list and return the value in the 1
[1] 6
```

# Selecting Data (cont'd)

- In the **tidyverse** we use **select()** to choose columns
- The **%>%** operator allows us to link steps together

```
1 speed <- read.table('file/cars.txt') %>%
2   select(., speed)
3 head(speed)
```

```
speed
1     4
2     4
3     7
4     7
5     8
6     9
```

# Selecting Data (cont'd)

- Use `slice` to get rows based on position

```
1 speed2 <- read.table('file/cars.txt') %>%
2   select(., speed) %>%
3   slice(., 2))  
  
speed  
2     4
```

# Selecting Data (cont'd)

- Use **filter** to choose rows that meet a condition

```
1 speed2 <- read.table('file/cars.txt') %>%
2   filter(., speed > 20))
```

	speed	dist
44	22	66
45	23	54
46	24	70
47	24	92
48	24	93
49	24	120
50	25	85

# Changing Data

- Updating data (CAUTION)
- Often using a combination of index and logicals

```
1 x <- runif(10,-10, 10) #generate 10 random numbers between -10 and 10
2 x[c(3,6,8)] <- NA #set the 3rd, 6th, and 8th value to NA
3 is.na(x) #check which values are NA
```

  

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

# Changing Data

- Creating new variables
- Can use \$

```
1 head(mtcars, 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

```
1 mtcars$hpwt <- mtcars$hp / mtcars$wt  
2 head(mtcars[,c(1, 5:12)], 3)
```

	mpg	drat	wt	qsec	vs	am	gear	carb	hpwt
Mazda RX4	21.0	3.90	2.620	16.46	0	1	4	4	41.98473
Mazda RX4 Wag	21.0	3.90	2.875	17.02	0	1	4	4	38.26087
Datsun 710	22.8	3.85	2.320	18.61	1	1	4	1	40.08621

# Changing Data

- Creating new variables
- Using **tidyverse, mutate** creates new variables for the entire dataset

```
1 mtcars_update <- mtcars %>%
2   mutate(., hpwt = hp/wt)
3 head(mtcars_update[,c(1, 5:12)], 3)
```

	mpg	drat	wt	qsec	vs	am	gear	carb	hpwt
Mazda RX4	21.0	3.90	2.620	16.46	0	1	4	4	41.98473
Mazda RX4 Wag	21.0	3.90	2.875	17.02	0	1	4	4	38.26087
Datsun 710	22.8	3.85	2.320	18.61	1	1	4	1	40.08621

# Changing Data

- Creating new variables
- Using **summarise** creates group level summaries

```
1 mtcars_group <- mtcars %>%
  2   group_by(., cyl) %>%
  3     summarise(., meanmpg = mean(mpg))
  4 mtcars_group

# A tibble: 3 × 2
  cyl  meanmpg
  <dbl>    <dbl>
1     4     26.7
2     6     19.7
3     8     15.1
```

# Getting help

# 2 Kinds of Errors

- **Syntax Errors:** Your code won't actually run
- **Semantic Errors:** Your code runs without error, but the result is unexpected

# Asking good questions

- What are you trying to do?
- What isn't working?
- What are you expecting?
- Why aren't common solutions working?

# Reproducible examples

- Don't require someone to have your data or your computer
- Minimal amount of information and code to reproduce your error
- Includes both code and your operating environment info
- See the **reprex** package.

# Wrap-up

