Data Objects and Visualization

Fall 2022

September 16 2022

Object Oriented Programming

In R, commands care about object class and type

- Ex: the default `plot` command wants a vector of data or a formula to form a scatterplot

```
plot(y ~ x, data= mydata) # makes scatterplot if x and y numeric
```

But if you give `plot` a `lm` regression object it will produce a set of diagnostic plots for that regression model.

```
my_lm <- lm(y ~ x, data= mydata)  # make a linear model
plot(my_lm)  # makes multiple diagnostic plots
```

Data structures and types in R

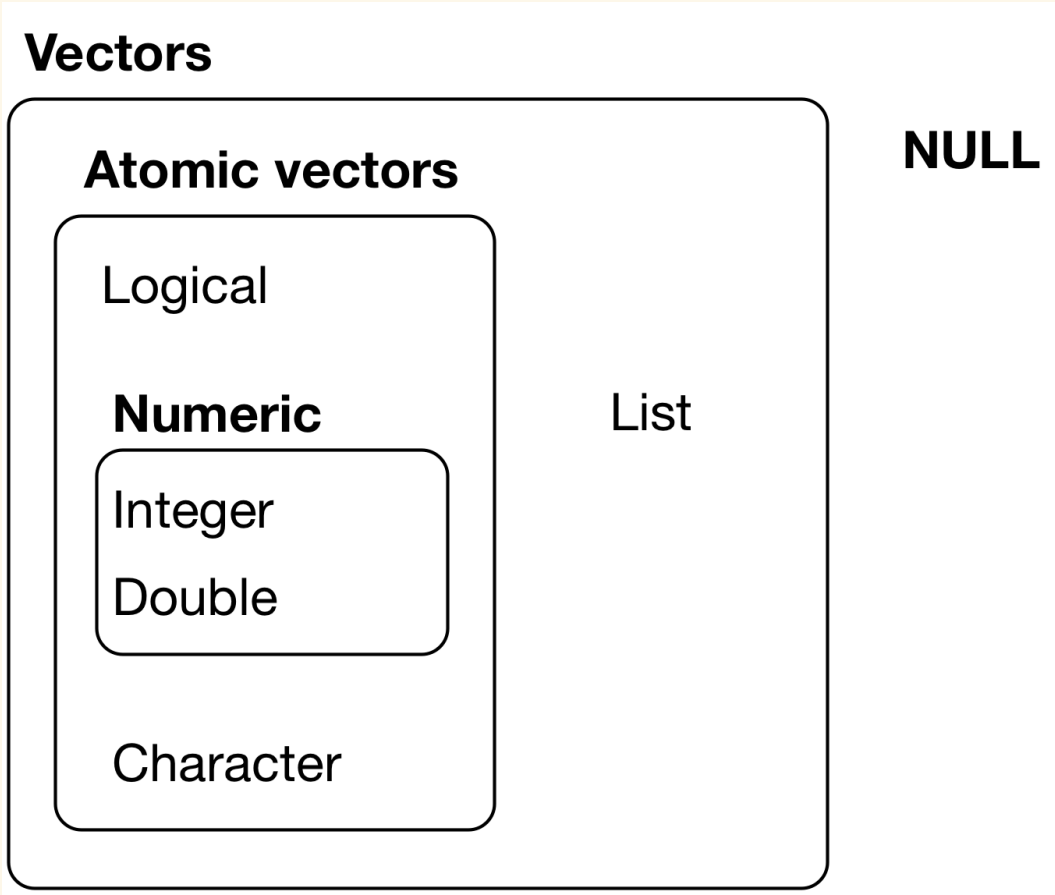
- Every object is a vector
- `NULL` = empty object (vector of length 0)
  - `typeof()`: tells us about storage of data
  - `class()` further describes the object

```
x <- c(8,2,1,3)
typeof(x)  # type of storage mode
[1] "double"
typeof(c(8L,2L,1L,3L))  # adding L forces
[1] "integer"
```

```
x == 1
[1] FALSE FALSE  TRUE FALSE
typeof(x == 1)
[1] "logical"
class(x)   # object class is numeric
[1] "numeric"
```

Atomic Vectors and lists

- R uses two types of vectors to store info
  - atomic vectors: all entries have the same data type
  - lists: entries can contain other objects that can differ in data type



Atomic Vectors: Matrices

- You can add attributes, such as dimension, to vectors
- A matrix is a 2-dimensional vector containing entries of the same type

```
x_mat <- matrix(x, nrow = 2, byrow = TRUE)
x_mat
     [,1] [,2]
[1,]    8    2
[2,]    1    3
attributes(x_mat)
$dim
[1] 2 2
```

```
typeof(x_mat)   # type of entries
[1] "double"
class(x_mat)   # info about object
[1] "matrix" "array"
```

- or you can bind vectors of the same length to create columns or rows:

```
x_mat2 <- cbind(x, 2*x)
x_mat2
     x
[1,] 8 16
[2,] 2  4
[3,] 1  2
[4,] 3  6
```

Implicit Coercion

- Entries in atomic vectors must be the same data type
- R will default to the most complex data type if more than one type is given

```
y <- c(1, 2, "a")
```

```
typeof(y)
[1] "character"
y
[1] "1" "2" "a"
```

Explicit coercion

- Intentionally forces a data type that is different from the "default" type

```
y <- as.character(c(1,2,3))
```

```
typeof(y)
[1] "character"
y
[1] "1" "2" "3"
```

Logical Vectors coercion

- Logical values coerced into 0 for `FALSE` and 1 for `TRUE` when applying math functions

```
x <- c(8,2,1,3)
x >= 5  # which entries >= 5?
[1]  TRUE FALSE FALSE FALSE
sum(x >= 5)  # how many >=5 ?
[1] 1
```

- What will `mean` of a logical vector measure?

```
mean(x >= 5)
```

```
[1] 0.25
```

Data types: factors

- Factors are a class of data that are stored as integers

```
x_fct <- as.factor(c("yes", "no", "no"))
class(x_fct)
[1] "factor"
typeof(x_fct)
[1] "integer"
```

- The attribute `levels` is a character vector of possible values
  - Values are stored as the integers (1=first `level`, 2=second `level`, etc)
  - Levels are ordered alphabetically/numerically (unless specified otherwise)

```
str(x_fct)
 Factor w/ 2 levels "no","yes": 2 1 1
levels(x_fct)
[1] "no"  "yes"
```

Subsetting: Atomic Vector and Matrices

- subset with `[]` by referencing index value (from 1 to vector length):

```
x
[1] 8 2 1 3
x[c(4, 2)]  # get 4th and 2nd entries
[1] 3 2
```

- subset by omitting entries

```
x[-c(4, 2)]  # omit 4th and 2nd entries
[1] 8 1
```

- subset with a logical vector

```
# get 1st and 3rd entries
x[c(TRUE, FALSE, TRUE, FALSE)]
[1] 8 1
```

- access entries using subsetting `[row,column]`

```
x_mat2
     x
[1,] 8 16
[2,] 2  4
[3,] 1  2
[4,] 3  6
```

```
x_mat2[, 1] # first column
[1] 8 2 1 3
```

```
x_mat2[1:2, 1] # first 2 rows of first column
[1] 8 2
```

R doesn't always preserve class:

```
# one row (or col) is no longer a matrix (1D)
class(x_mat2[1,])
[1] "numeric"
```

Subsetting: Atomic Vector and Matrices

- you can access entries like a matrix:

```
x_df <- data.frame(x = x, double_x = x*2)
x_df
  x double_x
1 8       16
2 2        4
3 1        2
4 3        6
```

```
x_df[, 1]  # first column, all rows
[1] 8 2 1 3
```

- or access columns with `$`

```
x_df$x  # get variable x column
[1] 8 2 1 3
```

```
# first column is no longer a dataframe
class(x_df[, 1])
[1] "numeric"
```

Subsetting: Data frames or Tibbles

Tibbles

- are a new modern data frame
- never changes the input data types
- can have columns that are lists
- can have non-standard variable names
  - can start with a number or contain spaces

- Can also use column names to subset:

```
library(babynames)
# get 2 rows of Name and Sex
babynames[1:2, c("name", "sex")]
# A tibble: 2 × 2
  name  sex
  <chr> <chr>
1 Mary  F
2 Anna  F
```

Subsetting lists

- Recall: a list is a vector with entries that can be different object types

```
my_list <- list(myVec = x,
                myDf = x_df,
                myString = c("hi", "bye"))
my_list
$myVec
[1] 8 2 1 3

$myDf
  x double_x
1 8        16
2 2         4
3 1         2
4 3         6

$myString
[1] "hi"  "bye"
```

- Like a data frame, can use the `$` to access named objects stored in the list

```
my_list$myDf
  x double_x
1 8        16
2 2         4
3 1         2
4 3         6
```

```
class(my_list$myDf)
[1] "data.frame"
```

Subsetting lists

- one `[]` operator gives you the object at the given location but preserves the list type
  - `my_list[2]` return a list of length one with entry `myDf`

```
my_list[2]
$myDf
  x double_x
1 8       16
2 2        4
3 1        2
4 3        6
```

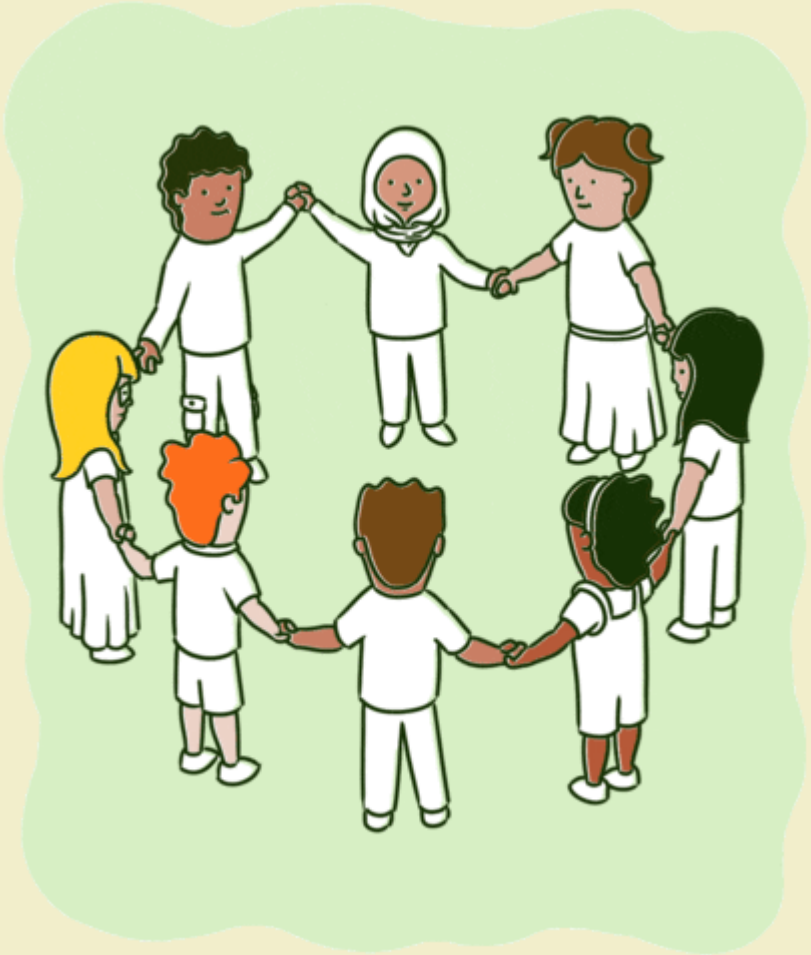
```
str(my_list[2])
List of 1
 $ myDf:'data.frame':    4 obs. of  2 variables:
  ..$ x       : num [1:4] 8 2 1 3
  ..$ double_x: num [1:4] 16 4 2 6
```

- the double `[[]]` operator gives you the object stored at that location (equivalent to using `$`)
  - `my_list[[2]]` or `my_list[["myDf"]]` return the data frame `myDf`

```
my_list[[2]]
  x double_x
1 8       16
2 2        4
3 1        2
4 3        6
str(my_list[[2]])
'data.frame':    4 obs. of  2 variables:
 $ x       : num  8 2 1 3
 $ double_x: num  16 4 2 6
```

Group Activity 1

10:00



- Let's go over to maize server/ local Rstudio and our class moodle
- Get the class activity 3 file
- Please work on Problems 1-3
- Ask me questions
- Any Github related questions??

Data Visualization provides a powerful way to communicate data-driven findings, motivate analyses, and detect flaw

ggplot2 — Overview

- A powerful package for visualising data
- Used widely by academics and industries alike

Some useful resources

- The package documentation
- The book by its creator Hadley Wickham
- The reference page
- The extensions, maintained by the `ggplot2` community

ggplot2 — Basics

- The `ggplot` function and the `data` argument
  - specify a data frame in the main `ggplot` function

```
ggplot(data = df)
```

- The mapping aesthetics, or `aes`; most importantly, the variable(s) that we want to plot
  - specify as an additional argument in the same `ggplot` function

```
ggplot(data = df, mapping = aes(x = x-variable, y = y-variable))
```

ggplot2 — Basics

- The geometric objects, or geom; the visual representations
  - specify, after a plus sign +, as an additional function

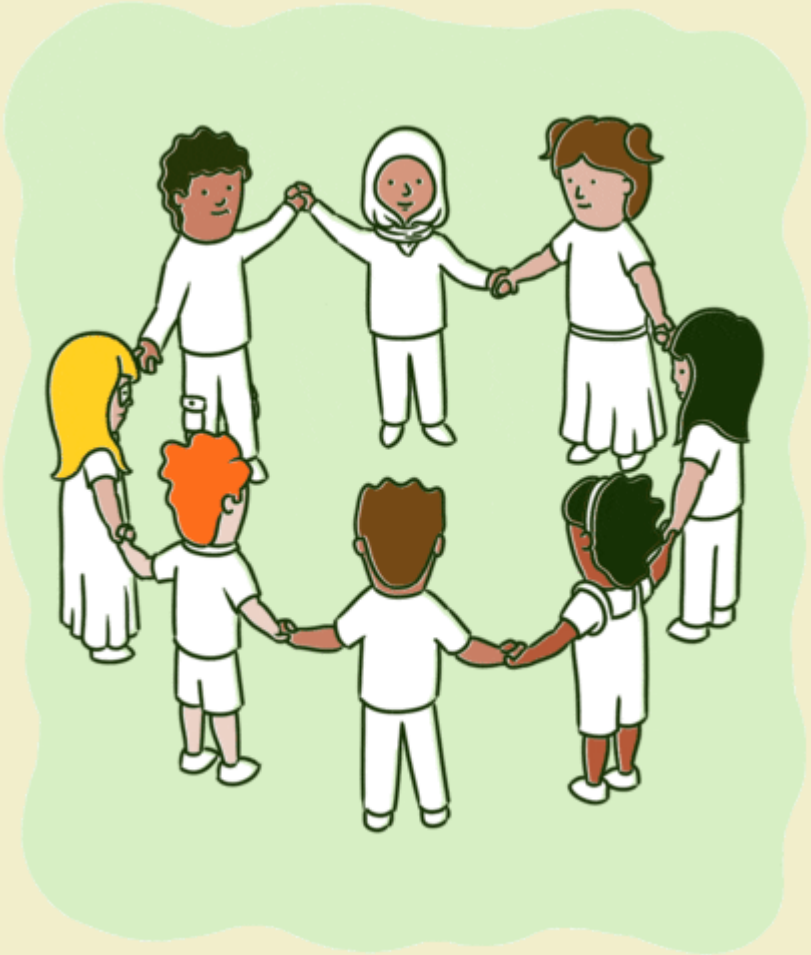
```
ggplot(data = df, mapping = aes(x = x-variable, y = y-variable)) +
      geom_point()
```

- Additional aesthestics like `color`, `size`, `shape`, and `alpha` (i.e. transparency) are possible.

```
ggplot(data = df, mapping = aes(x = x-variable, y = y-variable)) +
      geom_point(color = z-variable)
```

Group Activity 2

10:00



- Continue working on Problems 4-5
- Ask me questions
- Any more Github setup questions?