# Lecture 2 - Data Structures

Will Calandra

# Why Are We Learning Data Structures?

- R is convenient and R is smart, but sometimes R can be difficult. Therefore, you have to understand how functions work on different data structures to make R work for you instead of against you!
- Data cleaning - it's an essential part of the analytics workflow. Real world data tends to be messy, so it is imperative that you master these skills before you get into deep and complex programming!
- Syntax - the syntax R uses works well with its data structures, because arguments to functions are pulled from a certain "class" (which will be discussed later)!
- Speed - the last thing you should consider for now, but still ultimately important. Working with faster data structures makes larger computations easier and improves application performance!

# Classes

## Logical

### Overview

- T for TRUE and F for FALSE (both work, we prefer shorter for style)
  - Notice that this must be capitalized
  - Notice syntax highlighting (nice to use an IDE!)
- Used frequently to "turn on" or "turn off" options in functions

```
# Load R dataset
data <- mtcars

### Logical - either evaluates to true or false, is foundational for building computer logic
## T or F
# Use $ to access vectors
data$mpg == T
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
data$mpg > 15
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
## [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
```

### Logical operators

- ==
  - The most important operator - checks for equality, not assignment
  - Accidentally assigning with = when you meant == is one of the most common syntactical errors
- !=
  - Not equal to - you can get creative with this one! A lot of binary uses here
- “|”
  - The OR operator for conditionals (without the quotes)!
- &
  - The AND operator for conditionals!
- xor(x, y)
  - The OR operator but don’t include both!
- Click this link below for a nice summary graphic:
  - https://r4ds.had.co.nz/transform.html?q=near#logical-operators (https://r4ds.had.co.nz/transform.html?q=near#logical-operators)

```
## Operators - a bunch of different operators we use for comparison!
data$is0 <- data$vs == 0
data$is_not_0 <- data$vs != 0
data$mpg_15_17 <- data$mpg > 15 & data$mpg < 17
data$cyl_6_or_8 <- data$cyl == 6 | data$cyl == 8
data$xorTest <- xor(data$mpg_15_17 == T, data$cyl_6_or_8 == T) # function for xor (exactly one o
f these cases is true)
```

# Numeric

## Overview

- Just numbers (that’s it!)

## Double and integer

- There are some special cases for numbers - doubles have decimal places, integers don’t
- R will usually convert integers to doubles for you as needed (ha, C++)
- Why use integers if we have doubles? For memory: integers are fast and hold less data

## Mathematical Operators

- +, -, /, *
  - Addition, subtraction, division, multiplication
- ^ or **
  - Either works for exponential!
- %/% and %%
  - Integer division (5 %/% 2 is 2)
  - Remainders (5 %% 2 is 1)

## Relational Operators

- <, >, >=, <=
  - This is not assignment!
  - Less than, greater than, and the or equal to cases

```
## Mathematical Operators
# A bunch of different symbols which represent math operations!
data$cyl
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

```
data$cyl * 2
```

```
##  [1] 12 12  8 12 16 12 16  8  8 12 12 16 16 16 16 16 16  8  8  8  8 16 16 16 16
## [26]  8  8  8 16 12 16  8
```

```
data$cyl + 2
```

```
##  [1]  8  8  6  8 10  8 10  6  6  8  8 10 10 10 10 10 10  6  6  6  6 10 10 10 10
## [26]  6  6  6 10  8 10  6
```

```
data$cyl - 2
```

```
##  [1] 4 4 2 4 6 4 6 2 2 4 4 6 6 6 6 6 6 2 2 2 2 6 6 6 6 2 2 2 6 4 6 2
```

```
data$cyl / 2
```

```
##  [1] 3 3 2 3 4 3 4 2 2 3 3 4 4 4 4 4 4 2 2 2 2 4 4 4 4 2 2 2 4 3 4 2
```

```
data$cyl ^ 2
```

```
##  [1] 36 36 16 36 64 36 64 16 16 36 36 64 64 64 64 64 64 16 16 16 16 64 64 64 64
## [26] 16 16 16 64 36 64 16
```

```
data$cyl ** 2
```

```
##  [1] 36 36 16 36 64 36 64 16 16 36 36 64 64 64 64 64 64 16 16 16 16 64 64 64 64
## [26] 16 16 16 64 36 64 16
```

```
# Integer division - floors extra decimals
data$cyl %/% 2
```

```
##  [1] 3 3 2 3 4 3 4 2 2 3 3 4 4 4 4 4 4 2 2 2 2 4 4 4 4 2 2 2 4 3 4 2
```

```
# Modulus - remainder after division - handy to break up integers into pieces in calculations + build logic
data$cyl %% 3
```

```
##  [1] 0 0 1 0 2 0 2 1 1 0 0 2 2 2 2 2 2 1 1 1 1 2 2 2 2 1 1 1 2 0 2 1
```

```
## Relational Operators (greater than, less than, equal to cases, etc)
data$mpg > 17.3
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [25]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
```

```
data$mpg >= 17.3
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
## [13]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [25]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
```

# Character

## Brief Overview

- String
    - String transformations - regex! There are a ton of functions out there to help out (gsub() is a common one)
- Look out for “1”, “T”, etc.
    - They can be sneaky characters when you import data!
- Lots of data cleaning is getting data out of character form and into numeric form

```
### Character
# Load R dataset
data2 <- iris
# Click on "data2" and hover over the column headers - R gives you the data type (or class)!

# This is a "factor" data type - useful for grouping qualitative variables in analyses
print(data2$Species)
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```
# How to overwrite the Species column to change it from a factor data type to a character data t
ype
data2$Species <- as.character(data2$Species)

# Notice the "" marks - character coercion worked!
print(data2$Species)
```

```
##   [1] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##   [6] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [11] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [16] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [21] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [26] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [31] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [36] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [41] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [46] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [51] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [56] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [61] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [66] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [71] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [76] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [81] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [86] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [91] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [96] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
## [101] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [106] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [111] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [116] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [121] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [126] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [131] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [136] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [141] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [146] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
```

```
# gsub() in action - overwrite virginica to rose
data2$Species_Change <- gsub(pattern = "virginica",
                             replacement = "rose",
                             x = data2$Species)
data2$Species_Change
```

```
##   [1] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##   [6] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [11] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [16] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [21] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [26] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [31] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [36] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [41] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [46] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [51] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [56] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [61] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [66] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [71] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [76] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [81] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [86] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [91] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [96] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
## [101] "rose"       "rose"       "rose"       "rose"       "rose"
## [106] "rose"       "rose"       "rose"       "rose"       "rose"
## [111] "rose"       "rose"       "rose"       "rose"       "rose"
## [116] "rose"       "rose"       "rose"       "rose"       "rose"
## [121] "rose"       "rose"       "rose"       "rose"       "rose"
## [126] "rose"       "rose"       "rose"       "rose"       "rose"
## [131] "rose"       "rose"       "rose"       "rose"       "rose"
## [136] "rose"       "rose"       "rose"       "rose"       "rose"
## [141] "rose"       "rose"       "rose"       "rose"       "rose"
## [146] "rose"       "rose"       "rose"       "rose"       "rose"
```

# Special Cases

- NULL
  - Don’t worry about this too much, it just means empty data. R doesn’t like this, so you may have to change these to NA
  - Use is.null() to determine if something is NULL
  - Note NULL is in caps
- NA
  - The standard way of representing missing data
  - Note that NA == NA is not always true (interesting class case)
  - So use is.na() to determine whether data is NA
  - Note NA is in caps
- NaN, Inf
  - You violated some math rules (i.e. divide by 0)

```
### Special cases
data$hp <- NULL # removes column
data$qsec <- NA # makes the column empty
5 / 0 # infinity
```

```
## [1] Inf
```

```
0 / 0 # not a number
```

```
## [1] NaN
```

# (Atomic) Vectors

## Overview

- R is all about vectors
  - It is the fundamental data structure of R
- One class
  - class() - to check a class of a vector
- Any length, no negatives
  - length() - to check the length of a vector
- Let's learn our first few functions, without formal structure yet
  - c(), seq(), rep()
- Everything is a vector in R. They sound scary from math class, but they are foundational for data analysis!

## Indexing

- Use vec[] to call index of vectors
- Index search

## Filtering

- Use vec[] to filter vectors

## Functions and Operators are Nearly All Vectorized!

- This is the central concept behind why R is such a nice language to work with for data science
- Makes your life a lot easier
- No looping like other languages

## Coercion and Messy Vectors

- as. can help you coerce to the class you want!
- You may lose data by doing this, so watch for errors or warnings
- Some messy vectors:
  - x <- c(2, T)
  - x <- c(2, "hello")
  - x <- c("hello", 3.5)

## Recycling

- Beware of recycling
  - Recycling vectors of length one is obvious (adding 1 to every entry)
  - Recycling longer vectors gets messy
    - We did some of this in our first lecture - remember how R reads! Top to bottom, and you can overwrite in your script - but be careful in tracking your memory!

## Some Statistical Functions to Call on Vectors

- mean(), sum(), var()
- Implicit coercion from logical to numeric

```
### Vectors
class(data) # class() function checks the data type/class --- dataframes are foundational to R - they are like excel sheets! Rows and columns of data
```

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

```
length(data) # 15 columns in this dataframe
```

```
## [1] 15
```

```
class(data2) # also a dataframe
```

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

```
x <- c("Hey", "What's", "Up") # spawn a vector of three entries
class(x) # check it's type
```

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

```
# Use seq, rep functions to generate a vector of 150 entries (the y vector repeated 50 times)
y <- seq(from = 1, to = 5)
z <- rep(y, 30)

z[30] # find the 30th entry in the z vector
```

```
## [1] 5
```

```
z_slice <- z[20:40] # subset and create a new z_slice vector (comes from the 20th through 40th (inclusive) entries of z)
z_slice <- as.data.frame(z_slice) # coerces the vector into a dataframe so we can see it!
z_slice
```

```
##    z_slice
## 1        5
## 2        1
## 3        2
## 4        3
## 5        4
## 6        5
## 7        1
## 8        2
## 9        3
## 10       4
## 11       5
## 12       1
## 13       2
## 14       3
## 15       4
## 16       5
## 17       1
## 18       2
## 19       3
## 20       4
## 21       5
```

```
t <- rep(x, 40) # same process of rep, just on x vector, which is character data

# Difference?
x <- 1
y <- c("1")
class(x)
```

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

```
class(y)
```

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

```
# The data types are different!

# Messy
x <- c(2, T)
x <- c(2, "hello")
t <- c("hello", 3.5)
# These are really bad to work with, different data types in the same vector
# Use filters to get one data type per column, and then coerce to the proper class

# Fix
x <- c(2, "2")
x <- as.numeric(x) # R is smart here - can read that "2" as a number, even though it isn't
t <- as.numeric(t) # Not so much here - can't convert characters to numbers
```

```
## Warning: NAs introduced by coercion
```

```
is.na(t) # useful function to find NAs in vectors
```

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

```
# Vector Functions
mean(z)
```

```
## [1] 3
```

```
sum(z)
```

```
## [1] 450
```

```
var(z)
```

```
## [1] 2.013423
```

# This here below is for your reference (will come in handy for lectures 3 + 4)

## Dataframes/Tibbles

## Overview

- A dataframe is just a bunch of vectors lined up next to one another in columns. The gold standard package for operating on dataframes is called dplyr. It is written and maintained by Hadley Wickham, Chief Data Scientist at RStudio.
- Install the dplyr package:

```
install.packages("dplyr")
library(dplyr)
```

- Read the gold standard textbook on dplyr below:
    - https://r4ds.had.co.nz/transform.html (https://r4ds.had.co.nz/transform.html)

## dplyr Functions

- Syntax: data first, no need to repeat df name! We'll find practical uses in our next lecture…
- mutate()
- filter()
- select()
- arrange()
- summarize()
- group_by()