

# Week 2: Data Objects

01/21/2020

Jake Campbell

# Packages

- R has a huge library of packages that expand on base R functions
- Some packages are installed with your installation of R
  - We can load these with the `library()` function

```
library(MASS)
```

- Most packages we'll use need to be installed first using `install.packages()`
- We only need to install a package once, but we need to load it up every new R session

# Data Types

- Data can take on many different types of values
  - Data is not just numbers!
  - What about text, images, etc.?

# Major Data Types

- Numeric: number values

```
class(100)
```

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

- Character: text strings specified by quotation marks

```
class("Jake")
```

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

# Major Data Types

- Factor: categorical data

```
class(factor(c("second", "first"), levels = c("first", "second")))
```

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

- Logical: boolean true or false
  - Specified either with all capital letters (**TRUE**) or by just the first letter (**T**)

```
class(FALSE)
```

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

# Major Data Types

- Not a specific data type, but **NA** represents missing values in R
- We can't perform operations on it directly because it is not known

```
1 + 2 + NA
```

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

# Factors

- Factors are for categorical data rather than just any string data
- Categorical data should have a limited amount of possible options
  - There are only so many months in a year, but an infinite number of names
- Factors have a set text input, but also underlying levels specifying possible input and their order

```
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",  
                  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")  
month_factor <- factor(c("Dec", "Aug", "Oct"), levels = month_levels)  
month_factor
```

```
## [1] Dec Aug Oct  
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

# Factors

- Remember that factors are DIFFERENT than characters
- If we want to perform text manipulation, we need our data to be in character format
- A lot of models in R require data to be in factor form rather than character
  - A lot of the time, character data gets turned into factors automatically by the modeling function



# Changing Between Data Types

- To change between data types, we specify the data type we want to change to, prefaced by **as.**

```
# Changing from numeric to character  
test_character <- as.character(100)  
class(test_character)
```

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

```
# Changing from character to numeric  
test_number <- as.numeric(test_character)  
class(test_number)
```

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

# Changing Between Data Types

- The only hiccup you might experience is going from factor to numeric
- Calling `as.numeric` on a factor will give you the level order rather than the expected labels

```
test_factor <- factor(c(100, 200, 300), levels = c(200, 300, 100))  
test_factor
```

```
## [1] 100 200 300  
## Levels: 200 300 100
```

```
as.numeric(test_factor)
```

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

# Changing Between Data Types

- We would instead need to wrap the factor in `as.character` first before `as.numeric`

```
as.numeric(as.character(test_factor))
```

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

# Data Objects

- So we know data types, but what do we store the data in???
- R has different types of data objects
  - Each have their own requirements and presentations of data

# Vectors

- Vectors are just a simple combination of data of the same type
  - `c` is the combine function

```
test_vector <- c(1:10)
test_vector
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

# Matrices

- Matrices can hold two dimensions of data of the same type
  - Rows and columns
- Computationally fast and usable for matrix applications

```
# We have to specify the number of rows and columns in a matrix
test_matrix <- matrix(data = 1:9, nrow = 3, ncol = 3)
test_matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

# Lists

- Lists can contain data of different types
- They can be a bit confusing to work with

```
test_list <- list(name = c("Jake", "Hannah", "Jim"),  
                  age = c(26, 26, 33))  
test_list
```

```
## $name  
## [1] "Jake" "Hannah" "Jim"  
##  
## $age  
## [1] 26 26 33
```

# Data Frames

- Data frames are going to be the type of object we work with most
- Multi-dimensional, like a matrix, but can hold multiple data types
- Resembles a spreadsheet in appearance

```
test_df <- data.frame(name = c("Jake", "Hannah", "Jim"),  
                      age = c(26, 26, 33))  
  
test_df
```

```
##      name age  
## 1   Jake  26  
## 2 Hannah  26  
## 3    Jim  33
```



# Indexing Objects

- What if we want the 3rd observation from a vector, or the 5th column of a data frame?
  - We can index these in multiple ways
- Brackets allow us to specify element, row, or column of a data object

```
test_vector[3]
```

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

# Indexing Objects

- Indexing the data frame requires a comma because it has two dimensions
  - Put a 1 in front of the comma and it returns the first row
  - Put a 1 after the comma and it returns the first column

```
test_df[1, ]
```

```
##   name age  
## 1 Jake  25
```

```
test_df[, 1]
```

```
## [1] Jake   Hannah Jim  
## Levels: Hannah Jake Jim
```

# Indexing Objects

- Certain objects can also be indexed by the \$ followed by the column name we want to select

```
test_df$age
```

```
## [1] 25 26 33
```

# Function Syntax

- We can write our own functions to use later on in R
- We first specify the `function()` function
  - Its arguments are the names of the arguments we'll include in our function
- Next, we create curly braces `{ }`
  - Everything inside our curly braces is the body of the function
  - What is saved in the curly braces isn't saved in the global environment

```
test_function <- function(input) {  
  input + 1  
}  
  
test_function(1)
```

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