

# Teaching R in DASC 1104

## R Part II

John Tipton

The University of Arkansas

2021/06/07 (updated: 2021-06-07)

# Learning R

- Introduction to tidy statistics
- RStudio primers
- Software carpentries
  - R for Reproducible Analyses
  - Programming with R
- R for data science

# Materials

- Available on gitHub at XXXXXX

# Course make file

- In the loaded `dasc1104-teaching` project, the following command will generate all code

```
# note: this will take a while, maybe 15-20 minutes  
source(here::here("make.R"))
```

## Lectures

- `unit_4_09_intro_R_Rmarkdown.Rmd`
- `unit_4_R_10_intro_plotting.Rmd`
- `unit_4_R_11_File_IO_and_basic_data_wrangling.Rmd`
- `unit_4_R_12_Data_types_and_objects.Rmd`
- `unit_4_R_13_Functional_programming.Rmd`
- `unit_4_R_14_Intro_to_modeling.Rmd`

# Brief recap of each unit

# Intro to R and Rmarkdown

- Focus on reproducibility and workflow
- Heavily inspired by [R for Reproducible Analyses](#)
- Much of this was covered in part I

# Data visualization and plotting

- Goal: to get students interested and excited when working with data
  - Give enough background and language for students to seek out help online for data visualization
  - Emphasis on flexibility and exploration of data through visualization
  - Worked through a couple datasets to demonstrate how the visualization process works

# Data Visualization

- Using `ggplot` and the `mtcars` data, generate a scatterplot of fuel milage (`mpg`) as a function of engine power (`hp`)
- Explore how other variables in the `mtcars` dataset influence fuel milage



# File IO

- Read the `game_goals.csv` file from the `data` folder with `read_csv()` (from the `tidyverse` library) and give this `data.frame` the name `dat`
- Make sure you properly define the filepath
- Many more specifics about reading files
  - skipping lines
  - defining column variable types
  - different file delimiters
  - missing values and `NAs`
  - data comments

# Data manipulation

Most important `dp` `lyr` (a part of the `tidyverse`) commands

- Using pipes `%>%`
- Choose observations (rows) based on conditional values with `filter()`
- Reorder the variables with `arrange()`
- Select variables by name with `select()`
- Create new variables with `mutate()`
- Create summary variables with `summarize()`
  - `summarise()` if you are British
- Group variables with `group_by()`

# Pipes

The itsy bitsy spider went up the water spout  
Down came the rain and washed the spider out  
Out came the sun and dried up all the rain  
And the itsy bitsy spider went up the spout again

- hard to read

```
went(  
  sun(  
    rain_down(  
      went(itsy_bitsy_spider, up = "waterspout"),  
      washed = "out"),  
    dried = "rain"),  
  up = "spout again"  
)
```

- Much better

```
itsy_bitsy_spider %>%  
  went(up = "waterspout") %>%  
  rain_down(washed = "out") %>%  
  sun(dried = "rain") %>%  
  went(up = "spout again")
```

# Pipes

- How does the pipe work?

```
x <- 1:10  
x %>%  
  mean()
```

```
## [1] 5.5
```

is equivalent to

```
x %>%  
  mean(.)
```

```
## [1] 5.5
```

where the `.` is a placeholder for the variable `x` which results in

```
mean(x)
```

```
## [1] 5.5
```

# Pipes

- Pipes are not always the optimal choice
- Try to avoid super long chains of pipes
  - Instead: break up the pipe into intermediate objects
- Pipes can't handle multiple inputs/outputs
- How to build a chain of pipes
  - Start by outlining a plan of attack
  - Write one small subset of the pipe chain at a time and verify the results
- Challenge: Ask me questions about the `diamonds` data

# Data manipulation

- Using the NHL goals dataset from before (named `dat`), `filter()` out all records where the `player` is "Alex Ovechkin" or "Sidney Crosby". How many observations are there?
- Using the NHL goals dataset, `arrange()` the observations in decreasing order (the `desc()` function) using the `penalty_min` variable. Who on the list has two games in the top 10 penalty minutes of the dataset?
- Using the NHL dataset, `select()` the variables `player`, `shots`, and `goals`. Then, `mutate()` a new variable `shots_per_goal` by dividing the number of `shots` by the number of `goals`. Did you get any warnings/errors?
- Using the NHL dataset, `summarize()` the total `goals` for each `player`. Using `arrange()`, who are the top 5 players in all time goals (in the data)?

# Relational data

- Joining multiple datasets together
- `left_join()`
- `right_join()`
- `inner_join()`
- `outer_join()`

# Processing strings

- Use the `stringr` package (part of the `tidyverse`)
- Can use regular expressions

```
(string1 <- "This is a string")
```

```
## [1] "This is a string"
```

```
(string2 <- 'This is also a string')
```

```
## [1] "This is also a string"
```

```
(string3 <- 'This is a "quoted" string')
```

```
## [1] "This is a \"quoted\" string"
```



# String processing

- `str_length()`
- `str_c()`
- `str_sub()`
- others

# Factors

- Factors are an atomic type for categorical data
  - By default, these are unordered
  - Ordered factors can be used (good - better - best)
- `factor()`
- `levels()`
- `fct_recode()`
- `fct_order()`

# Dates (not covered)

# Functional programming (currently not covered, but should be)

- Code is for **humans** not computers
- Functions make code readable to humans

# Keep it DRY

- Don't Repeat Yourself
- If you copy and paste code 3 or more times, write a function
- Generate some data

```
df <- data.frame(  
  x = rnorm(10),  
  y = rnorm(10),  
  z = rnorm(10)  
)
```

- The above isn't DRY, but there is no chance of errors and the code is easy to understand

# DRY

- Transform each variable to mean 0, standard deviation 1

```
x_scaled <- (df$x - mean(df$x, na.rm = TRUE)) / sd(df$x, na.rm = TRUE)
y_scaled <- (df$y - mean(df$y, na.rm = TRUE)) / sd(df$y, na.rm = TRUE)
z_scaled <- (df$z - mean(df$z, na.rm = TRUE)) / sd(df$y, na.rm = TRUE)
```

- Can you spot the error?

```
z_scaled <- (df$z - mean(df$z, na.rm = TRUE)) / sd(`df$y`, na.rm = TRUE)
```

- write a function to do this

```
rescale_variable <- function(x) {
  mean_x <- mean(x, na.rm = TRUE)
  sd_x    <- sd(x, na.rm = TRUE)
  ## save the variable for the output
  out <- (x - mean_x) / sd_x
  return(out)
}
```

- Much less likely to make an error

```
x_scaled <- rescale_variable(df$x)
y_scaled <- rescale_variable(df$x)
z_scaled <- rescale_variable(df$z)
```

# Naming Functions

- Name functions for human readers!

```
# bad name -- what does this do?  
f()  
  
# what does this do?  
func()  
  
# longer names -- use autocomplete  
fill_missing_values()  
pre_process_data()
```

- Don't name your functions over functions or variables which already exist!

```
mean <- function(x) {}  
TRUE <- function(x, y) {}
```

# Naming Functions

- Pick a naming convention and stick with it (I prefer snake case)

```
# snake case  
my_function_name <- function(x, y) {}  
# camel case  
myFunctionName <- function(x, y) {}
```

- Try to reuse leading names -- allows for autocomplete

```
# better  
check_input()  
check_parameters()  
check_inits()  
  
# not as good  
input_check()  
parameters_check()  
inits_check()
```

---

## For loops

- for loops in R



# Writing functions

- given a `data.frame`, calculate the mean of each column
- write a function that returns the mean of each column

# The project

- Idea: have each student construct a blog and perform a basic analysis using the skills learned in the class