Module 2

Day 1

Recap last week

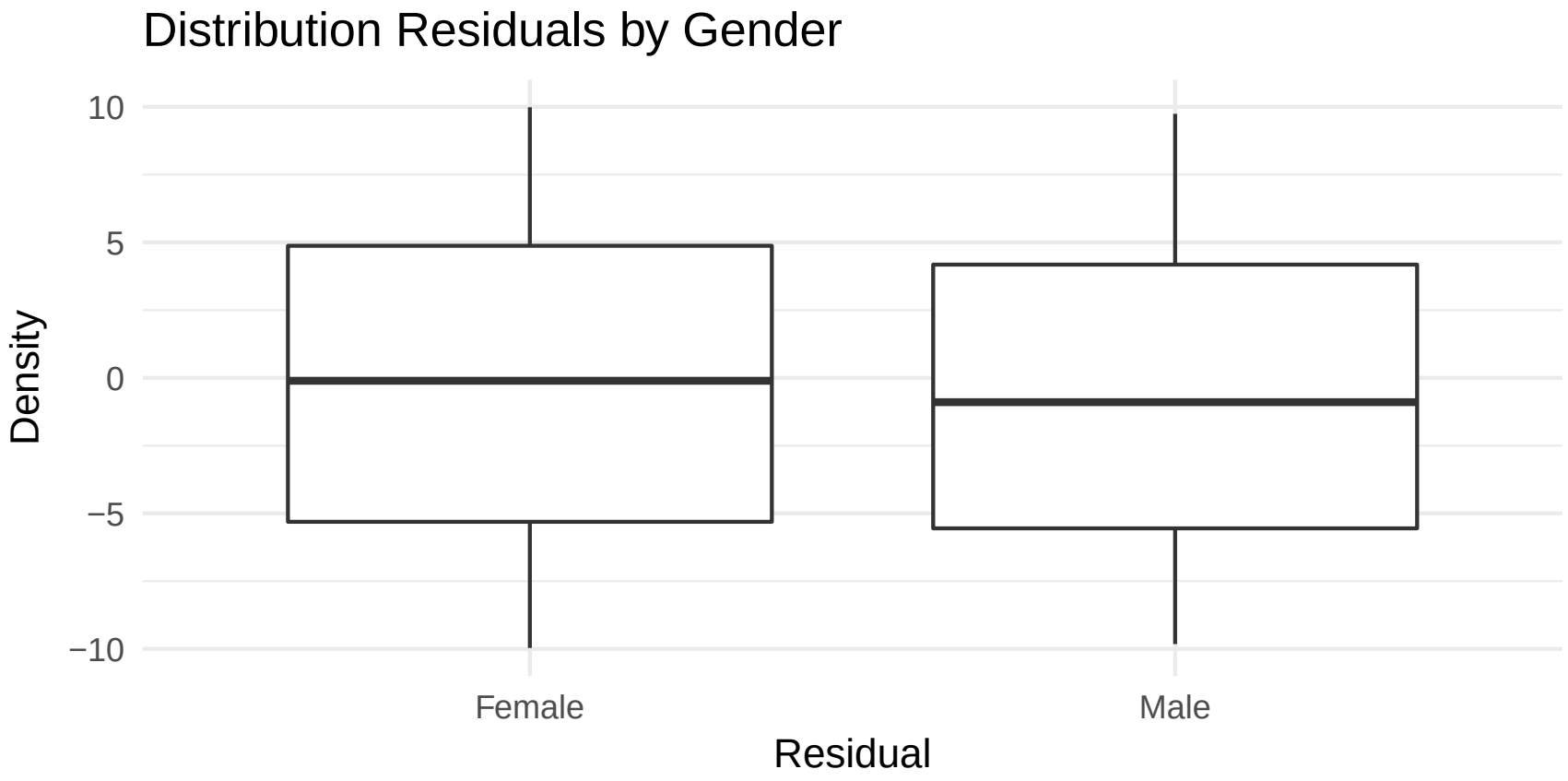
- Define and discuss when to use regression analysis
- Create two regression models with `lm()`:
  - 1. A simple linear regression including weights
  - 2. Add dummy variables
- Use `stargazer()` to view then interpret regression coefficients
- Check quality of our models with residuals and adjusted R^2
- Introduce the `augment()`, `tidy()`, and `gather()` functions

Recap last week

Table 1: Model Comparison

	hourly_wage	
	(1)	(2)
age	3.697***	3.701***
	(0.159)	(0.159)
sexMale		53.058***
		(4.188)
Constant	56.727***	28.541***
	(6.824)	(7.150)
Observations	18,765	18,765
R^2	0.028	0.036
Adjusted R^2	0.028	0.036
Note:	*p<0.1; **p<0.05; ***p<0.01	

Recap last week



Goals for Today

R:

- Motivate why we need automation
- Learn how to use for loops and if/else statements
- Create code to read in multiple data sets automatically

Economics:

- Acquaint ourselves with property data provided by Redfin

Redfin Data

- Seattle-based online retail company
- Website includes basic information such as:
  - Price
  - Square footage
  - Year built
  - List and Sale Date
  - Location (including longitude and latitude)
- All houses in the DC Metro area for 3 months prior to October 19, 2017

Redfin Data

- However we have two issues even before we begin cleaning:
  1. Redfin only allows downloads of 350 properties or less at a time
  2. Location and sale data are separate
- We will need a way to easily read in multiple data sets

Checking the data

```
##  [1] "acs_2016_cleaned.csv"   "location_redfin_01.csv"
##  [3] "location_redfin_02.csv" "location_redfin_03.csv"
##  [5] "location_redfin_04.csv" "location_redfin_05.csv"
##  [7] "location_redfin_06.csv" "location_redfin_07.csv"
##  [9] "location_redfin_08.csv" "Metro_lat_lon.xlsx"
## [11] "property_redfin_01.csv" "property_redfin_02.csv"
## [13] "property_redfin_03.csv" "property_redfin_04.csv"
## [15] "property_redfin_05.csv" "property_redfin_06.csv"
## [17] "property_redfin_07.csv" "property_redfin_08.csv"
```

Previewing the property data

- Let's take a peak at the property data

```
property_data <- read_csv("Data/property_redfin_01.csv")
names(property_data)
```

- We have some variables that look like they could be very useful in our analysis
- Let's look at the data directly using `View()`
  - Are there any columns that concern you?
  - Are there any columns you feel we should drop?

```
View(property_data)
```

In-Class Exercise

What is the maximum square footage of a condo in property_data?

Previewing the location data

Read in one of our location files, what are the names of the columns in the file? What does this data look like? Are there any columns that exist in both our property and location data?

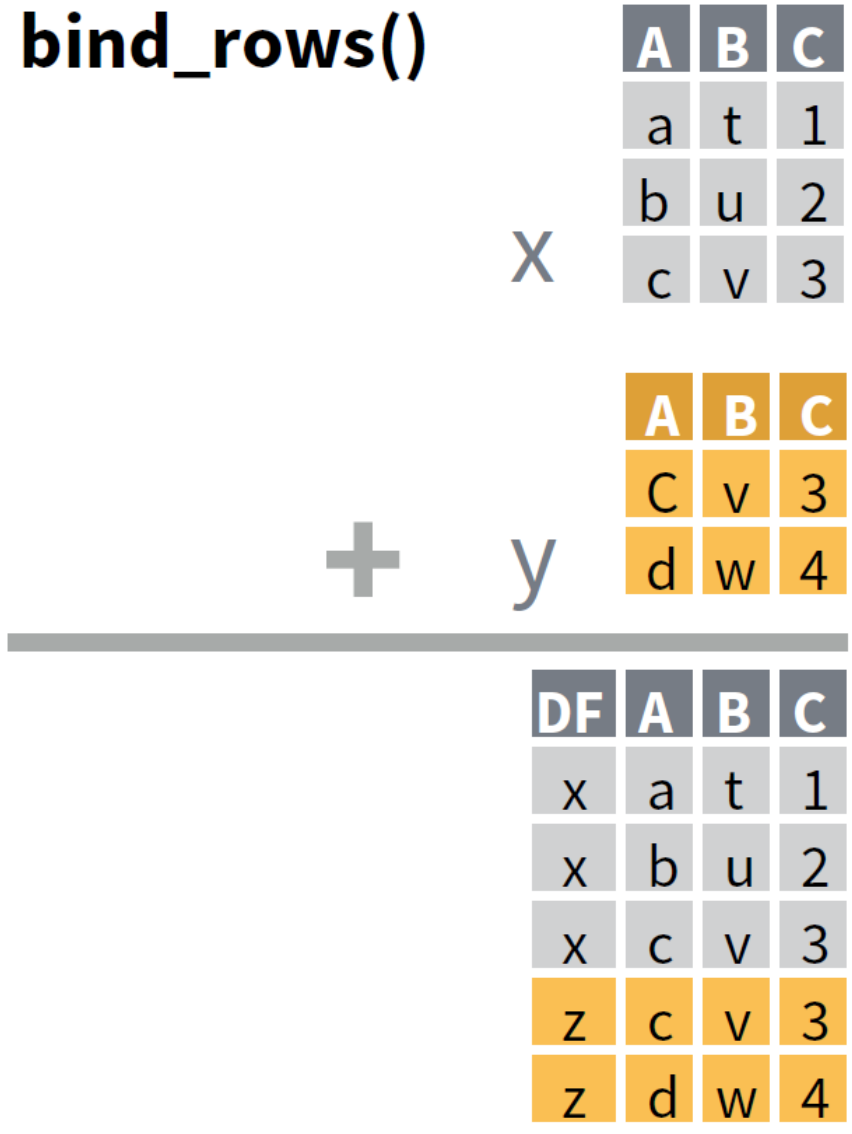
Combining property data

- We can infer these data sets belong together, but we should always double check:

```
property_data2 <- read_csv("Data/property_redfin_02.csv")
identical(names(property_data2), names(property_data))
names(property_data2)
```

- All of our column names are exactly the same!
- We can use a function called `bind_rows()` to stack the data on top of one another

Combining property data



Combining property data

- For data sets with identical variable names, use `bind_rows()` when we wish to combine observations from each set into one table.
- As long as the data have the same variable names, they do not have to be in the same order
- Now let's take this to the code:

```
nrow(property_data)
nrow(property_data2)

property_data_bound <- bind_rows(property_data,
                                 property_data2)
nrow(property_data_bound)
```

Combining location data

Are our location files also the same? Read in a second location file and test if it has the same columns as the first one. If it does, create a new table, location_data_bound which has the data for both tables in it.

Automating our code

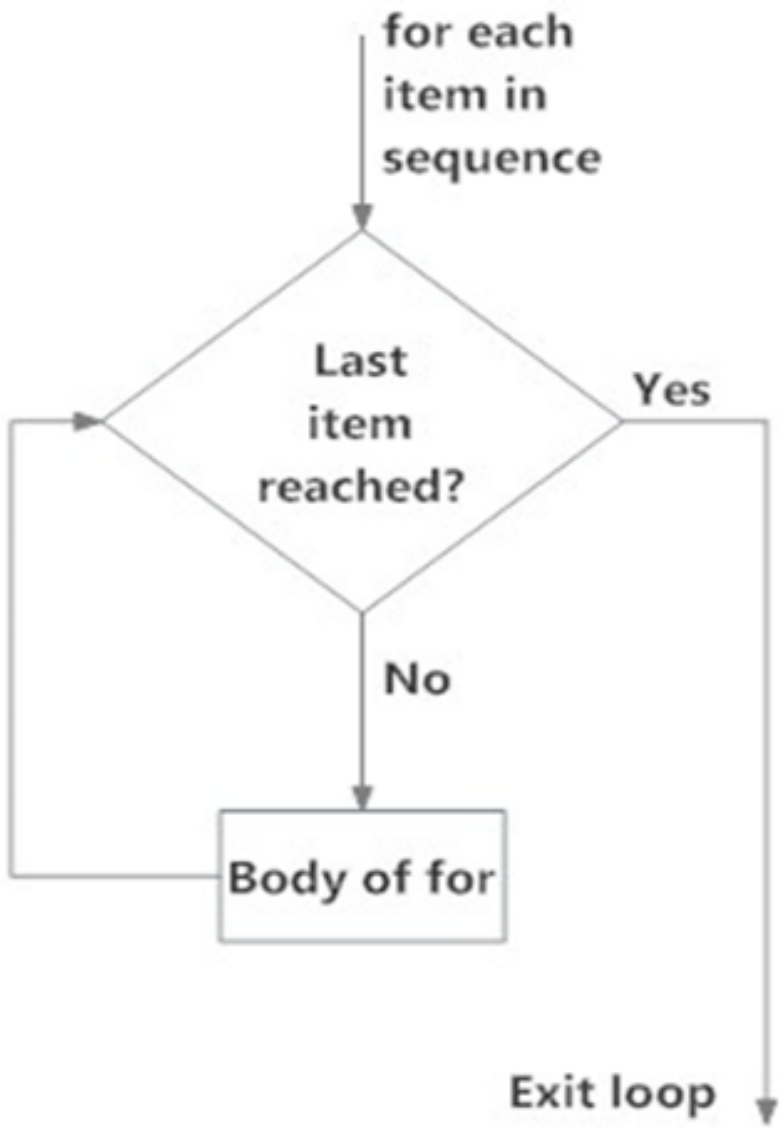
- Often when we prepare data for analysis, we have to repeat a task or calculation multiple times
  - Still have to add 6 more sets of data to `property_data_bound` and `location_data_bound`!
- Not practical or efficient to do these tasks manually
  - Time consuming — What if we wanted real estate data for the last year? Or the last 10 years?
  - Could introduce mistakes that are hard to find
  - Makes our code less readable
- For loops are one way we can automate tasks in R

for loops

- For loops have three key components:
  1. A variable that is used within the code of the loop
  2. A set of numbers or elements that the loop iterates over
  3. The code that is to be executed in the loop
- In R, for loops take the following form:

```
for(index in set) {
    ## code to be executed
}
```

Anatomy of a for loops



Simple example

- Let's square every number from 1 to 8 and print the results

```
for(num in seq(1, 8)){
    print(num^2)
}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
## [1] 64
```

Another simple example

- For loops can iterate over lists and vectors as well
- Let's try printing all of the months of the year contained

```
months <- c("January","February", "March", "April",
            "May", "June", "July", "August",
            "September", "October","November",
            "December")

for(mon in months){
  print(mon)
}

for(mon in months){
  print(paste(mon, "is my favorite month."))
}
```

- Is there another way that we can loop over elements of a vector using numbers?

Using indices to loop

```r
pets <- c("platypus", "frog", "cat")

length(pets)

#Printing indices
for(pet in seq(1, length(pets))){
  print(pet)
}

#Using indices to print elements
for(pet in seq(1, length(pets))){
  print(pets[pet])
}
```

Improving our for loops

- We can also save and update variables and data using for loops
  - first initialize variable or data frame we want **before the loop**
  - then append results to this object during the loop
  - make use of "helper" variables to prevent overwriting variable/data frame
- In our example we will be performing a cumulative sum
  - A running total where the current cumulative sum is equal to the sum of all previous elements and the current element

Cumulative sum **without** saving work

```r
num_vec <- seq(1,5)

for(currentNum in seq(1, length(num_vec))) {
  # Calculate sum at current iteration
  currentCumulSum <- sum(num_vec[1:currentNum])

  # Print current cumulative sum
  print(currentCumulSum)
}
```

Cumulative sum **with** results saved

```r
num_vec <- seq(1,5)

# Initialize object before the loop
cumulSum <- numeric()

for(currentNum in seq(1, length(num_vec))) {
  # Calculate sum at current iteration
  currentCumulSum <- sum(num_vec[1:currentNum])

  # Write to object during loop to record progress
  cumulSum <- c(cumulSum, currentCumulSum)
}

print(cumulSum)
```

```
## [1]  1  3  6 10 15
```

- Initialize the vector for our sequence cumulSum
- Calculate the next number in the sequence using our helper — `currentCumulSum`

In-Class Exercise: Creating a string from substrings

Now it's your turn! Try converting this vector of substrings into a single string. Make sure to use meaningful variable names in your code. (Hint: Use the paste function.)

```
string_vec <- c("Never","Gonna", "Give", "You", "Up",
                "Never", "Gonna", "Let", "You", "Down")

## initialize the variable

for(index in set){
  ## code to be executed
}
```

Vectorized operations vs. for loops

- Over the semester we have made ample use of vectorized operations
  - These are operations that take vectors as inputs and return a vector as output
  - An example: creating a variable in a data frame that is an elementwise sum of two other variables
    - `df %>% mutate(var3 = var1 + var2)`
    - When you sum `var1` and `var2` you get back a vector that is just as long - a sign you are doing elementwise operations
- We could instead build variables with the iterative for loop process, and at times that is our only choice. If you have any of the following then you likely need to use for loops to create objects:
  - a dependent, iterative process
  - functions with non-vector inputs
  - outputs with unknown output size even with known inputs

Improving our for loops

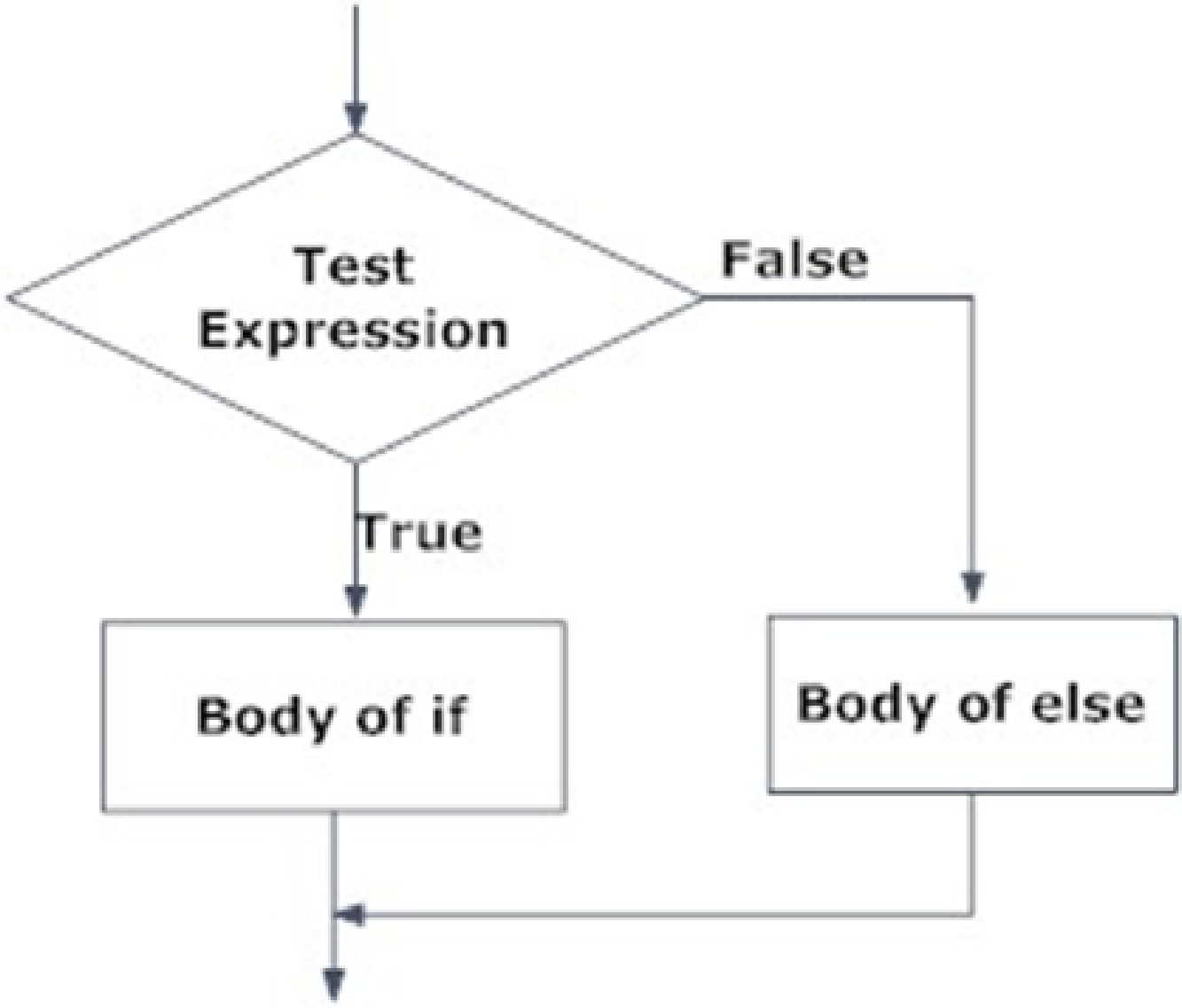
- Our loops are still less than ideal
- What if we want to apply one action on a certain part of our list and a different action on another part?
  - Using multiple loops can be messy and may require debugging
  - Want our code to make these decisions for us automatically
- We've already used two functions for this kind of decision making before:
  - `ifelse()`
  - `case_when()`
- But those functions are used *within* other functions

If Else Statements

- If Else statements are used directly in the code
  - Can't be implemented inside of premade functions
- In R, a basic if else statement takes the following form:

```
if(logical argument){
  ## code to be executed
} else{
  ## code to be executed
}
```

Anatomy of an if else statement



Evens and Odds

- We will use the modulo operator (%%) to characterize numbers as even or odd
- %% return the remainder after division

```
# %% is the modulus operator
# We are finding the remainder!
4 %% 1
10 %% 4
3 %% 2
```

```
testNum <- 12
if(testNum %% 2 == 0){
  print(paste(testNum, "is EVEN"))
} else{
  # If the remainder after division by 2 is not 0
  # then it must be odd (right..?)
  print(paste(testNum, "is ODD"))
}
```

Evens and Odds

- Let's characterize and record the numbers from 1 to 5 as even or odd

```
evens <- numeric()
odds <- numeric()

for(i in seq(1, 5)){
  if(i %% 2 == 0){ # %% is the modulus operator
                   # --- we are finding the remainder!
    evens <- c(evens, i)
  } else{
    odds <- c(odds, i)
  }
}
```

```
evens
```

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

```
odds
```

```
## [1] 1 3 5
```

In-Class Exercise

Given the following grade, use an if else statement to determine if the student passed or failed (cutoff is 50, 50 being a pass)

```
student_grade <- 71
```

Return a correct statement that the student passed or failed

Else if statements

- We aren't limited to choosing between two conditions
- Similar to `case_when()`, else if lets us make multiple decisions
  - provides us with even more flexibility
- Checks each condition one by one
  - check the first condition, if false it moves on to the the next one
- **Else catches everything that does not meet the previous criteria** so be careful when coding or deciding what to include

Assigning Grades

```
test_scores <- c(85,55,100,67,73,92,94,99,87)
# Initialize our vector
letter_grades <- NULL
for(grade in test_scores){
  if(grade >= 90){
    letter_grades <- paste(letter_grades,"A")
  } else if(grade >= 80) {
    letter_grades <- paste(letter_grades,"B")
  } else if (grade >= 70) {
    letter_grades <- paste(letter_grades,"C")
  } else if (grade >= 60) {
    letter_grades <- paste(letter_grades,"D")
  } else {
    letter_grades <- paste(letter_grades,"F")
  }
}
```

In-Class Excercise

Create a loop that will take the square root of a positive number or give us an NA if the number is negative. Save the results in a vector you initialized outside of the loop.

Reading in the Redfin Data

- Before we loop over our data sets we need a list of all of the data sets
- Luckily, our files have uniform names!
- Note on `paste`/`paste0`: Takes any number of strings, or vectors that can be coerced to character, and makes one string

```
## create a vector of data set names
property_files <- paste0("Data/property_redfin_0",
                         1:8, ".csv")
location_files <- paste0("Data/location_redfin_0",
                         1:8, ".csv")

# combine them into a single vector
files <- c(property_files, location_files)
files
```

str_detect()

- We can now load all of our data sets, but we still want our loop to decide which data sets to combine together
  - Need to know whether a data set is a location or property file
- `str_detect()` is a function in the tidyverse package that can tell whether a string contains a certain word or phrase
  - `TRUE` if the word is in the string
  - `FALSE` if the word is *not* in the string

```
str_detect("property_redfin_01.csv", "property") # TRUE
str_detect("location_redfin_01.csv", "property") # FALSE

str_detect(c("property_redfin_01.csv",
             "location_redfin_01.csv"),
           "location") # c(FALSE, TRUE)
```

Reading in the data

Using what we have learned in class today, read in the location and property data using for loops and if else statements. Within in the loop, combine all of the location data into one data set, and all of the property data into one data set.

```
property_data <- data_frame()
location_data <- data_frame()

for(file in files){
    if(str_detect()){
      data <- ## code to be executed
      property_data <- ## code to be executed
    } else if(str_detect()){
      data <- ## code to be executed
      location_data <- ## code to be executed
    }
}
```

Appendix

Improving our for loops

- Looping over elements
  - When you have output from a for loop that you append to an object each time, you might start worrying about inefficiency
    - The object gets rewritten in memory at each iteration
- Looping over indices
  - Results are usually the same length as inputs because we want to perform some operation on *each* element
  - Therefore we know the size of our output!
  - You can then initialize your object with a pre-specified size and assign iterative output to their respective elements in the output object