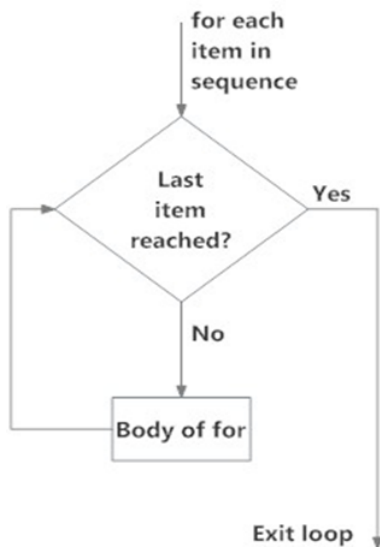


Module 2

March 9, 2018

Day 2

Recap Last Week



For-loops: Warm Up Exercise

- Cubing every number from 1 to 8 and print the results

```
for(num in      ){  
    print(      **3)  
}
```

For-loops: Warm Up Exercise (Answer)

```
for (num in seq(1, 8)) {  
  print(num^3)  
}
```

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

```
## [1] 8
```

```
## [1] 27
```

```
## [1] 64
```

```
## [1] 125
```

```
## [1] 216
```

```
## [1] 343
```

```
## [1] 512
```

Recap Last Week - In-Class Exercise

- ▶ Like we did last class, let's convert 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("Economics", "is", "the", "best",  
               "subject!")
```

```
## initialize the variable
```

```
for (item in string_vec) {  
  ## code to be executed  
}
```

Recap Last Week - In-Class Exercise (Answer)

```
string_vec <- c("Economics", "is", "the", "best",  
               "subject!")  
  
## initialize the variable  
phrase = NULL  
  
for (item in string_vec) {  
  phrase = paste(phrase, item)  
}  
  
print(phrase)
```

Goals for Today

Economics - Question of the Day

- ▶ How does the sale price of residential property change based on various factors?

What things will affect the property value?

Goals for Today

- ▶ Determinants of property values
 - ▶ Size of house (square footage)
 - ▶ Size of property
 - ▶ Higher floors vs lower floors
 - ▶ Quality of local schools
 - ▶ Number of bedrooms and bathrooms
 - ▶ Neighborhood location (metro, restaurants, library)

Goals for Today

Programming - R

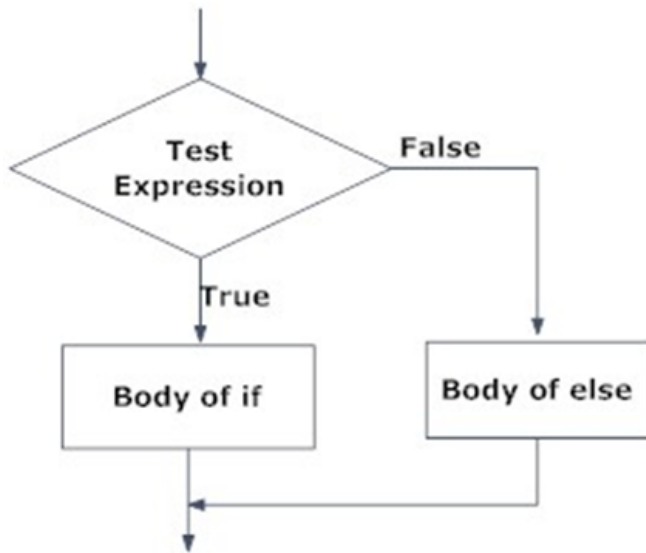
- ▶ Reviewing for-loops and if/else Statements
- ▶ Introduction to stringr
- ▶ Combining data using bindrows, %in%, and join functions
- ▶ Cleaning data
- ▶ Effective graphics for data visualization

If/Else Statements: Introduction

- ▶ We have previously used `ifelse()` to make decisions about recoding our variables with `mutate`
- ▶ If/else statements work in a similar way
- ▶ In R, a basic if/else statement takes the following form:

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

If/Else Statements: Example



Evens and Odds Example

- ▶ We will use the modulo operator (%%) to characterize numbers as even or odd
- ▶ %% returns the remainder after division of the first argument by the second argument

```
# %% is the modulus operator We are finding the  
# remainder!  
4%%1  
## [1] 0  
10%%4  
## [1] 2  
3%%2  
## [1] 1  
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"))  
}  
## [1] "12 is EVEN"
```

If/Else Statements Combined with For-Loops

- ▶ Let's characterize and record the numbers from 1 to 10 as even or odd

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

for (i in seq(1, 10, by = 1)) {
  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 6 8 10
```

```
odds
```

```
## [1] 1 3 5 7 9
```

In-Class Exercise: If/Else Statements

- ▶ Given the following grade, use an if/else statement to determine if the student passed or failed
 - ▶ **(A passing grade is greater than or equal to 60)**

```
student_grade <- 71
```

Return a correct statement that the student passed or failed

```
if(      ){  
  print("  ")  
} else {  
  print("  ")  
}
```

In-Class Exercise: If/Else Statements (Answer)

```
if (student_grade >= 60) {  
    print("Student passed")  
} else {  
    print("Student failed")  
}
```

```
## [1] "Student passed"
```


Else-if statements: Introduction

- ▶ 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 then 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

Else-if statements: Example

- Here's an example of assigning survey numerical responses to genders.

```
response_list <- c(0, 1, 2, 2, 0, 1, 2, 1, 0, 2,  
  2, 1, 1, 1, 2, 1, 0)  
female <- 0  
male <- 0  
other <- 0  
  
for (person in response_list) {  
  if (person == 1) {  
    male <- male + 1  
  } else if (person == 2) {  
    female <- female + 1  
  } else {  
    other <- other + 1  
  }  
}  
  
survey_genders <- data.frame(male, female, other)  
survey_genders
```

```
##   male female other  
## 1     7      6     4
```

Else-if statements: Assigning Grades - In-Class Exercise

- ▶ You've been given a list of test scores that you want to categorize into letter grades
- ▶ Use if else statements to assign letters to the numeric grades
 - ▶ A 90 - 100, B 80 - 89, C 70 - 79, D 60 - 69, F < 60

```
test_scores <- c(85,55,100,67,73,92,94,99,87,89.3)
```

```
# Initialize our vector
```

```
letter_grades <-
```

```
for(grade in test_scores){
```

```
  if(    >= 90){
```

```
    letter_grades <- paste(letter_grades,"A")
```

```
  } else if(    ) {
```

```
    letter_grades <-      (          , "B")
```

```
  } else if
```

```
    letter_grades <-
```

```
    else if
```

```
  } else {
```

```
    letter_grades <- paste(          "F")
```

```
  }
```

```
}
```

```
letter_grades
```

Else-if statements: Assigning Grades - In-Class Exercise (Answer)

```
test_scores <- c(85, 55, 100, 67, 73, 92, 94, 99,
  87, 89.3)
# 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")
  }
}
letter_grades
```

```
## [1] " B F A D C A A A B B"
```

Reading in Data: Introduction

- ▶ Before we can access our datasets we need to help R find them
- ▶ We can create a vector of all the files in our data folder using the `list.files` function
- ▶ **What sorts of files are in the folder?**

```
## create a vector of dataset names
temp_files <- list.files("./Data/")
temp_files
```

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

Reading in Data: Introduction

- ▶ We need to be able to access these files, we'll need to tell R to look in the folder 'Data'
 - ▶ Therefore, we should append 'data' onto the title using paste0

```
files <- paste0("./Data/", temp_files)
files
```

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

- ▶ Now that we know the names of the files with the datasets, we need to decide which datasets are useful and which go together
- ▶ **What are the two major categories of files?**
- ▶ **How can we combine these files most effectively?**

Introduction to str_detect()

- ▶ `str_detect()` is a function in the tidyverse package that can tell whether a string contains a certain word or phrase. The function returns:
 - ▶ TRUE if the word you're looking for is in the string
 - ▶ FALSE if the word you're looking for is *not* in the string

```
str_detect("property_redfin_01.csv", "property")
```

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

```
str_detect("location_redfin_01.csv", "property")
```

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

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

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

Using str_detect

- Using what we've learned about str_detect, let's create a vector that only includes location files.

```
files[str_detect(files, "location")]
```

```
## [1] "./Data/location_redfin_01.csv" "./Data/location_redfin_02.csv"  
## [3] "./Data/location_redfin_03.csv" "./Data/location_redfin_04.csv"  
## [5] "./Data/location_redfin_05.csv" "./Data/location_redfin_06.csv"  
## [7] "./Data/location_redfin_07.csv" "./Data/location_redfin_08.csv"
```


Building our Dataset - In-Class Exercise

- ▶ You will need to use for-loops and if-else statements to create two datasets, one dataset that includes all the 'location' data and one dataset that includes all the 'property' data.
 - ▶ Initialize two data frames to store all your new data: one data frame for the 'property data' and one for the 'location data'
 - ▶ Create a for-loop that cycles through each csv in the folder and checks if it is a 'location' or 'property file'.
 - ▶ If it's a property file, add (bind) the csv's data onto your property data frame
 - ▶ If it's a location file, bind it into your location_data data frame.

Hint(check out what the bind_rows function does.)

```
property_data <- data_frame() # initialize empty data frame for the property data
location_data <- data_frame() # initialize empty data frame for the location data
for(file in list.files(folder, pattern = "\\.csv$", full.names = TRUE)){
  # You only want to have property data in the first dataset
  if(str_detect(file, "property")){
    data <- read_csv(file)
    property_data <- bind_rows(property_data, data)
  }
  # Now let's combine location data into a different dataset
  else if(str_detect(file, "location")){
    data <- read_csv(file)
    location_data <- bind_rows(location_data, data)
  }
}
```

Building our Dataset - In-Class Exercise (Answer)

```
property_data <- data_frame() # initialize empty data frame
location_data <- data_frame() # initialize empty data frame

for(file in files){
  # this is a property file, add to property_data
  if(str_detect(file, "property")){
    data <- read_csv(file)
    property_data <- bind_rows(property_data, data)
  }
  # location file
  else if(str_detect(file, "location")){
    data <- read_csv(file)
    location_data <- bind_rows(location_data, data)
  }
}
```

Overview of the Data

- ▶ Let's take a look at the variables in our two dataframes - property data and location data
- ▶ **What does the property data do? What does the location data do?**
- ▶ The good news is that we have much of the data we need, the bad news is that the data is split between location information and price information
- ▶ **What piece of information is available in both datasets?**
- ▶ We want to combine these two datasets around that common piece of information

Comparing our Datasets

► Harnessing the power of the %in% function

```
vec1 <- c("green", "yellow", "dog", "teal", "violet",  
          "beige", "cat", "red", "horse", "orange")  
vec2 <- c("yellow", "red", "blue", "purple", "green",  
          "orange")  
# What does the %in% function show us?  
vec1 %in% vec2
```

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

```
# Where are the words that overlap across  
# vectors? Where are the words in vector 1 that  
# appear in vector 2?  
which(vec1 %in% vec2)
```

```
## [1] 1 2 8 10
```

```
# How do we return the names of the elements that  
# appear in both vectors?  
vec1[vec1 %in% vec2]
```

```
## [1] "green" "yellow" "red" "orange"
```

Where to Combine the Data

- ▶ Using our `%in%` function, let's find what variable(s) occur(s) in **both** our property and location datasets.
- ▶ We'll save the result as 'overlap', which is a variable we can use for our next step.

```
property_names <- names(property_data)
location_names <- names(location_data)

# What variable exists in both datasets?
property_names %in% location_names
```

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

```
overlap <- property_names[property_names %in% location_names]
overlap
```

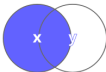
```
## [1] "URL"
```

“Joining” the Data: Introduction

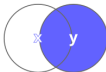
- ▶ Our URL column exists in both of the dataframes!
- ▶ We only want houses that exist in both datasets
- ▶ We call combining data frames “joining.” Below are types of joins:
- ▶ Which type of joining combines our dataframes so we only have properties with both location and property information?

dplyr *joins*

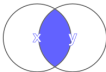
left_join(x, y)



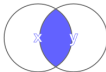
right_join(x, y)



inner_join(x, y)

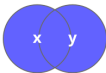


semi_join(x, y)

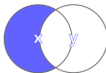


(never duplicate rows of x)

full_join(x, y)



anti_join(x, y)



“Joining” the Data: In Practice

```
joined_data <- inner_join(property_data, location_data,  
  by = overlap)
```

```
names(joined_data)
```

```
## [1] "SALE.TYPE"          "SOLD.DATE"  
## [3] "PROPERTY.TYPE"      "PRICE"  
## [5] "BEDS"               "BATHS"  
## [7] "SQUARE.FEET"        "LOT.SIZE"  
## [9] "YEAR.BUILT"         "HOA.MONTH"  
## [11] "STATUS"             "NEXT.OPEN.HOUSE.START.TIME"  
## [13] "NEXT.OPEN.HOUSE.END.TIME" "URL"  
## [15] "SOURCE"             "MLS."  
## [17] "FAVORITE"           "INTERESTED"  
## [19] "LIST.DATE"          "ADDRESS"  
## [21] "ZIP_CODE"           "LAT_LON"  
## [23] "CITY_STATE"         "LOCATION"
```

“Joining” the Data: Additional Notes

- ▶ Success! Now we have one dataset with all of our housing data
- ▶ Note - It is also possible to perform a `join` where the columns we are matching in each table do not have the same name using the ‘by’ argument: `by = c(“name1” = “name2”)`.
- ▶ Similarly, it is also possible to `join` based on multiple columns: `by = c(column1, column2, etc. . .)`

Examining the Data

- ▶ Now we can examine the data to check for any problems.
- ▶ **What are some problems you see in terms of usability for this data?**

Examining the Data: Potential Problems

- ▶ price column - USD in front of the numbers
- ▶ propertytype column - has “propertytype” in front of every observation
- ▶ city_state column - state is inconsistent (va, VA, Virginia) and unwanted states
- ▶ lat_lon column - & in the middle of the string
- ▶ zip_code column - extra zeros
- ▶ sold.dates, list.date columns - dates are not consistent

Cleaning the Data: Why We Do It

- ▶ If we try to make a plot without cleaning up our data...it doesn't work too well

`## Warning: Removed 13 rows containing missing values (geom`



Cleaning the Data: Price Column

- ▶ Let's start by looking at our price data more closely

```
head(joined_data$PRICE)
```

```
## [1] "USD280000" "USD405000" "USD510000" "USD395000" "USD339900" "USD415000"
```

- ▶ There's a USD in front of each number of the column
- ▶ What class of variable is the price column?

Cleaning the Data: Price Column

- ▶ How can we fix this problem for a character variable? What type of method could we use?

Cleaning the Data: Price Column

- ▶ One way we could eliminate the USD is by replacing the USD with nothing, "" (this would be the same as removing it).
- ▶ Try using the `str_replace` function to modify the string "USD40000" to be "40000"

```
str_replace("USD40000", "USD", "")
```

```
## [1] "40000"
```

Cleaning the Data: Price Column

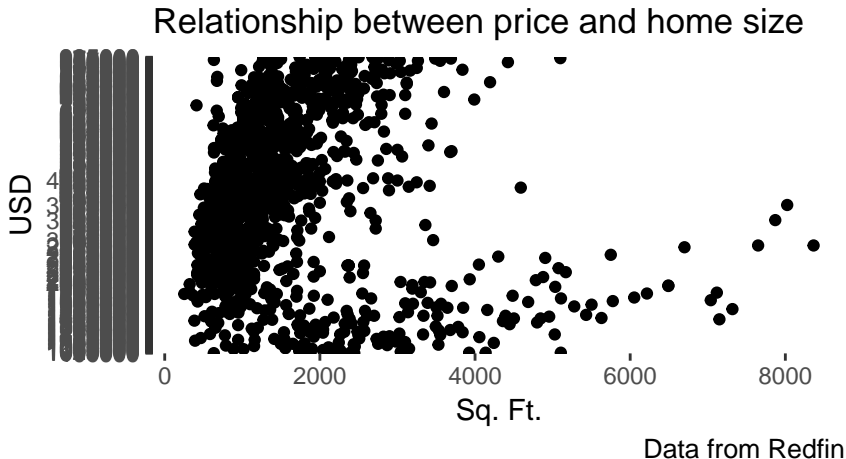
- ▶ Let's mutate the price column to remove "USD" from the character string

```
joined_data <- joined_data %>% mutate(PRICE = str_replace(PRICE,  
  "USD", ""))  
head(joined_data$PRICE)
```

Cleaning the Data: A Second Attempt at Plotting

- ▶ Let's redo the plot!

```
## Warning: Removed 13 rows containing missing values (geom
```



Cleaning the Data: Reclassifying Columns and Removing N/As - In-Class Exercise

- ▶ We still have a problem, we forgot to convert our PRICE column to be class numeric!
- ▶ We also currently have a warning about missing values – what should we do about those?
- ▶ You should:
 - ▶ Convert the price column to be numeric
 - ▶ Remove any missing values from the price and square feet columns
 - ▶ Then replot and see how it looks

Cleaning the Data: Reclassifying Columns and Removing N/As - In-Class Exercise

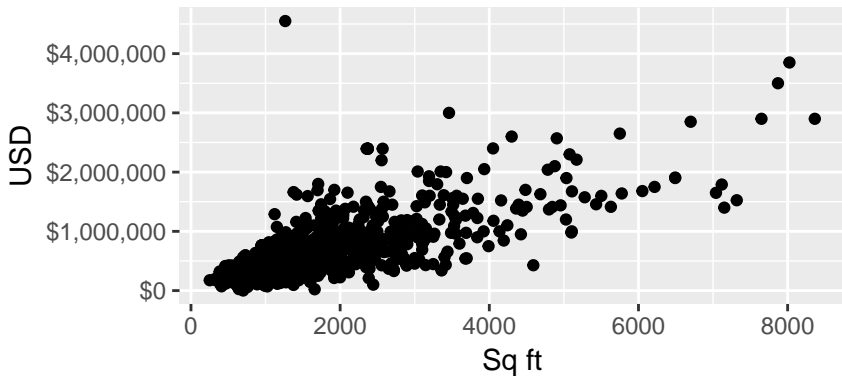
```
joined_data <- joined_data %>% filter(!is.na(), !is.na()) %>%  
  mutate(PRICE = as.numeric())  
  
# Now make the plot  
joined_data %>% ggplot(aes(x = , y = )) + geom_point() +  
  # Note that the scale_y_continuous makes sure we  
# don't get expotentials!  
scale_y_continuous("USD", labels = dollar) + labs(title = "Price vs. Home Size"  
  y = , x = , caption = "Data from Redfin")
```

Cleaning the Data: Reclassifying Columns and Removing N/As - In-Class Exercise (Answer - Code)

```
joined_data <- joined_data %>% filter(!is.na(PRICE),  
  !is.na(SQUARE.FEET)) %>% mutate(PRICE = as.numeric(PRICE))  
  
# Now make the plot  
joined_data %>% ggplot(aes(x = SQUARE.FEET, y = PRICE)) +  
  geom_point() + # Note that the scale_y_continuous makes sure we  
# don't get expotentials!  
scale_y_continuous("USD", labels = dollar) + labs(title = "Price vs. Home Size"  
  y = "USD", x = "Sq ft", caption = "Data from Redfin")
```

Cleaning the Data: Reclassifying Columns and Removing N/As - In-Class Exercise (Answer - Graph)

Relationship between price and home size



Data from Redfin

- What is the relationship between Sq Ft and home price?

Understanding the Data: Property Type and Price

- ▶ What about the relationship between property type (apt, single family house, condo, etc.) and price?
- ▶ Let's produce a table and a bar plot of the average price per square foot by property type.

Cleaning the Data: Property Type - In-Class Exercise

- ▶ Let's clean the property type column
- ▶ Check out the PROPERTY.TYPE column and use `str_replace` to fix it

```
joined_data <- %>%  
  mutate(PROPERTY.TYPE = str_replace(  
    "propertytype:",  ))
```

Cleaning the Data: Property Type - In-Class Exercise (Answer)

```
joined_data <- joined_data %>% mutate(PROPERTY.TYPE = str_replace(PROPERTY.TYPE  
  "propertytype:", ""))
```

Understanding the Data: Property Price/Sq Ft Table

- Let's create a table of average price per square foot by property type

```
type_price <- joined_data %>% group_by(PROPERTY.TYPE) %>%  
  summarise(price = mean(PRICE/SQUARE.FEET, na.rm = T))  
  
type_price
```

```
## # A tibble: 5 x 2  
##       PROPERTY.TYPE      price  
##       <chr>         <dbl>  
## 1      Condo/Co-op  456.6841  
## 2 Multi-Family (2-4 Unit) 295.3326  
## 3 Multi-Family (5+ Unit) 214.7378  
## 4 Single Family Residential 368.6735  
## 5      Townhouse    408.2464
```


Understanding the Data: Property Price/Sq Ft Graph

- ▶ Now let's check out a graph. This is obviously not a production-quality graph, just something for our reference.
- ▶ How are the colors defined in the graph?



Understanding the Data: Data Challenges

- City/state is another variable that might impact home sale price... Why?

```
unique(joined_data$CITY_STATE)
```

```
## [1] "Arlington, va"      "Alexandria, va"
## [3] "Arlington, VA"      "Falls Church, va"
## [5] "Arlington, Virginia" "McLean, Virginia"
## [7] "Chevy Chase, MD"    "Bethesda, MD"
## [9] "Glen Echo, MD"      "Kensington, MD"
## [11] "Washington, DC"     "Washington, Michigan"
## [13] "Oxon Hill, MD"      "Silver Spring, MD"
## [15] "Washington, COLORADO" "Capitol Heights, MD"
## [17] "Takoma Park, MD"    "Alexandria, VA"
## [19] "Fairmount Heights, MD"
```

- What issues do we have with the city_state variable?

Cleaning the Data: String Split

- ▶ We can use the `str_split_fixed` function to alter our string (or vector of strings) using a 'pattern'
- ▶ The function returns the peices of the string after splitting based on your 'pattern', which could be a word, symbol, or number of characters.

```
# We have three arguments, what do they do?  
strings <- str_split_fixed("United States of America",  
  "of", n = 2)  
strings
```

```
##      [,1]      [,2]  
## [1,] "United States " " America"
```

```
class(strings)
```

```
## [1] "matrix"
```

Cleaning the Data: String Split

- ▶ We told R to split into two pieces ($n = 2$), so we got two columns in return.
- ▶ If we want to get back to individual elements, we can subset the matrix using brackets.

```
strings[1, 1]  # first row, first column
```

```
## [1] "United States "
```

```
strings[1, 2]  # first row, second column
```

```
## [1] " America"
```

Cleaning the Data: String Split

- ▶ Let's test it out with something closer to our actual use case.
- ▶ Note that we're splitting along the comma and a space, which will remove both!

```
str_split_fixed("Arlington, Virginia", ",", " ", n = 2)
```

```
##           [,1]           [,2]  
## [1,] "Arlington" "Virginia"
```

Cleaning the Data: String Split - In-Class Exercise

- Use what we just learned to mutate the city_state column and create separate city and state columns.

```
joined_data <- %>%  
  mutate(CITY = str_split_fixed(CITY_STATE, " ")[1, 1],  
         STATE = (str_split_fixed(CITY_STATE, " ")[1, 2])  
  
joined_data %>% select(CITY_STATE, CITY, STATE) %>% head
```

Cleaning the Data: String Split - In-Class Exercise (Answer)

```
## # A tibble: 6 x 3
##   CITY_STATE      CITY STATE
##   <chr>      <chr> <chr>
## 1 Arlington, va Arlington  va
## 2 Arlington, va Arlington  va
## 3 Arlington, va Arlington  va
## 4 Arlington, va Arlington  va
## 5 Arlington, va Arlington  va
## 6 Arlington, va Arlington  va
```

Cleaning the Data: State Data

- ▶ We're not done yet. Let's take another look at the state column and see what types of responses exist

```
unique(joined_data$STATE)
```

```
## [1] "va"          "VA"          "Virginia" "MD"          "DC"          "Michigan"  
## [7] "COLORADO"
```

- * What do you think we need to do to further clean this column?

Cleaning the Data: Capitalizing States

- ▶ The function `str_to_upper` will convert all lower case letter to upper case ones

```
str_to_upper("va")
```

```
## [1] "VA"
```

- ▶ We'll use this function to mutate the state column to upper case

```
joined_data <- joined_data %>% mutate(STATE = str_to_upper(STATE))  
  
unique(joined_data$STATE)
```

```
## [1] "VA"          "VIRGINIA" "MD"        "DC"        "MICHIGAN" "COLORADO"
```

Cleaning the Data: Abbreviating States - In-Class Exercise

- ▶ Now we want to change “VIRGINIA” to “VA” in the column to make the label consistent
- ▶ We’ve already used this function! Go back and find the function you need to make that change

```
joined_data <- joined_data %>%  
  mutate(STATE = ( , "VIRGINIA", ))
```

Cleaning the Data: Abbreviating States - In-Class Exercise (Answer)

```
joined_data <- joined_data %>% mutate(STATE = str_replace(STATE,  
  "VIRGINIA", "VA"))
```

Cleaning the Data: Removing Additional States

- ▶ The last thing we need to do is toss out the rows with MICHIGAN and COLORADO since we only want DMV data
- ▶ Use the `%in%` function to filter only rows with VA, DC, or MD in them.

```
joined_data <- joined_data %>% filter(STATE %in%  
  c("VA", "DC", "MD"))  
  
unique(joined_data$STATE)  
## [1] "VA" "MD" "DC"
```

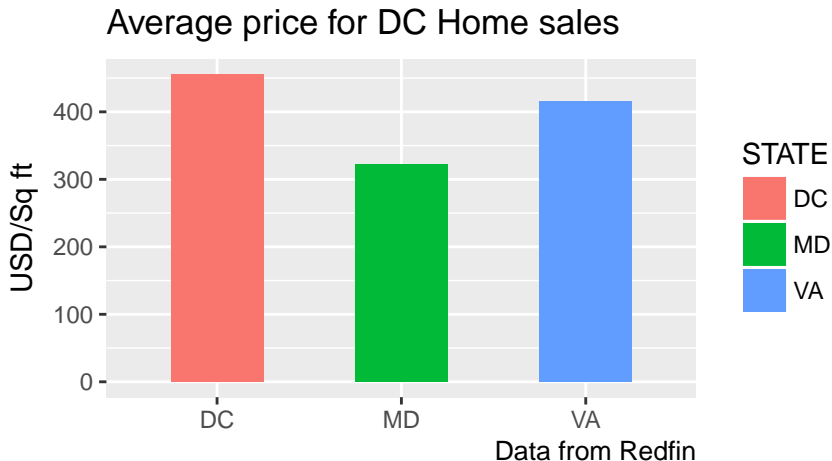
Plotting the Data: Price by State

- Let's find out the average prices by state for our dataset

```
state_average <- joined_data %>% group_by(STATE) %>%  
  summarise(price = mean(PRICE/SQUARE.FEET, na.rm = T))  
  
state_average_plot <- state_average %>%  
  ggplot(aes(x = STATE, y = price, fill = STATE)) +  
  geom_bar(stat = "identity", width = 0.5) +  
  labs(x = NULL, # What does this do?  
       y = "USD/Sq ft",  
       title = "Average price for DC Home sales",  
       caption = "Data from Redfin")  
  
state_average_plot
```

Plotting the Data: Price by State

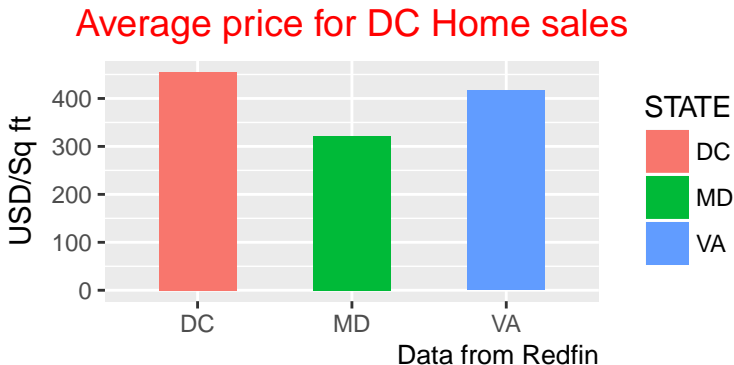
- ▶ I want to have a larger plot title and have it centered
- ▶ How can we do that? Would we use a data-related dimension or an aesthetic option?



Plotting the Data: Price by State

- ▶ We can use the `theme()` function to alter elements of the graph.
- ▶ What other textual elements can we alter in this graph?

```
state_average_plot + theme(plot.title = element_text(size = 14,  
  hjust = 0.5, color = "Red"))
```



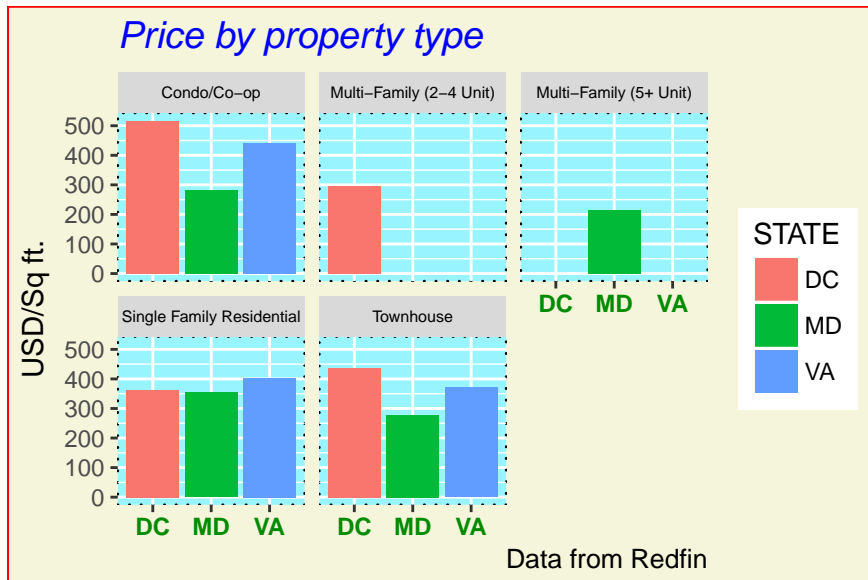
Plotting the Data: Appearance Options

- ▶ We can use `element_text` to control any text on our graph, including the title, axis labels, legend, etc. `element_text` allows us to set:
 - ▶ font family (Times New Roman, Arial, etc. . .)
 - ▶ font size (10, 12, 14, etc. . .)
 - ▶ font face (bold, italic)
 - ▶ color
 - ▶ `hjust` (horizontal adjustment)
 - ▶ `vjust` (vertical adjustment)
 - ▶ angle
 - ▶ other aspects that impact the display of text.

Plotting the Data: Property Type

[illegible]

Plotting the Data: Property Type



Understanding the Graphic Appearance

- ▶ We used `element_rect()` to adjust rectangular elements of the plot.
- ▶ The main plot has a beige background with a red border.
- ▶ We used the `panel.background` option to adjust elements of the area where the data is
- ▶ What did the `linetype` argument do in our `panel.background` option?

```
plot <- state_proptype %>% ggplot(aes(x = STATE,  
  y = price, fill = STATE)) + geom_bar(stat = "identity") +  
  facet_wrap("PROPERTY.TYPE") + labs(title = "Price by property type",  
  x = NULL, y = "USD/Sq ft.", caption = "Data from Redfin")  
  
plot + theme(plot.title = element_text(face = "italic",  
  size = 14, color = "Blue"), axis.text.x = element_text(face = "bold",  
  color = "green4"), strip.text.x = element_text(size = 6,  
  color = "black"), plot.background = element_rect(fill = "beige",  
  color = "red"), panel.background = element_rect(fill = "cadetblue1",  
  color = "black", linetype = 3))
```

Understanding the Data: Price per Lot Size

- ▶ Perhaps using the price per square foot is not the best method of analysis.
- ▶ When a property is sold the surrounding land around the house will likely affect the selling price.
- ▶ Let's work on making a plot of price/lot size

Understanding the Data: Price per Lot Size

- ▶ We'll start by looking at the head of the data

```
## # A tibble: 6 x 3
##   PRICE SQUARE.FEET LOT.SIZE
##   <dbl>         <int>    <int>
## 1 280000         1055      NA
## 2 405000         1030      NA
## 3 510000         1209      NA
## 4 395000         1135      NA
## 5 339900          930      NA
## 6 415000         1606      NA
```

- ▶ We have a lot of NA values. Why is that?

Understanding the Data: Price per Lot Size - In-Class Exercise

- ▶ Let's get rid of the NA values by replacing them with square footage
- ▶ We'll create a new column called Property Size that equals either lot size or square feet
 - ▶ Use an ifelse() statement in our mutate, where we test whether lot size is NA
 - ▶ If lot size is NA, use square feet. Otherwise, use lot size
- ▶ Next we will summarize our data by price per foot of property size.
 - ▶ We want to group by property type and state!

```
joined_data <-      %>%  
  mutate(PROPERTY.SIZE = ifelse(is.na(      ), SQUARE.FEET,      ))  
  
# Now we can find our average  
lot_state_data <- joined_data %>%  
  group_by(      ) %>%  
  summarise(price = mean(      , na.rm = T))  
  
head(lot_state_data)
```

Understanding the Data: Price per Lot Size - In-Class Exercise (Answer)

```
joined_data <- joined_data %>% mutate(PROPERTY.SIZE = ifelse(is.na(LOT.SIZE),  
  SQUARE.FEET, LOT.SIZE))
```

Now we can find our average

```
lot_state_data <- joined_data %>% group_by(PROPERTY.TYPE,  
  STATE) %>% summarise(price = mean(PRICE/PROPERTY.SIZE,  
  na.rm = T))
```

```
head(lot_state_data)
```

```
## # A tibble: 6 x 3
```

```
## # Groups:   PROPERTY.TYPE [4]
```

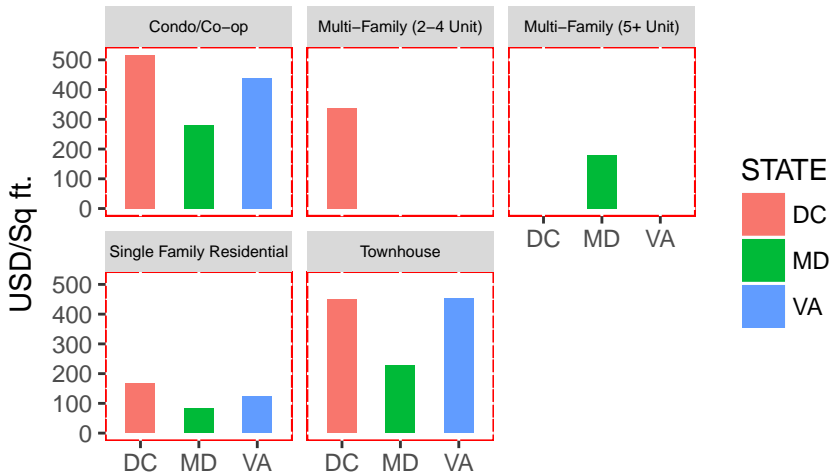
##	PROPERTY.TYPE	STATE	price
##	<chr>	<chr>	<dbl>
## 1	Condo/Co-op	DC	517.2998
## 2	Condo/Co-op	MD	281.4152
## 3	Condo/Co-op	VA	439.9952
## 4	Multi-Family (2-4 Unit)	DC	338.9963
## 5	Multi-Family (5+ Unit)	MD	179.8902
## 6	Single Family Residential	DC	166.2042

Understanding the Data: Culminating Exercise

- ▶ Make a plot like the one we just did but using our new price/property.size variable with some modifications -
 - ▶ **Make your title centered (using hjust) and green**
 - ▶ **Make your panel background white with a red border**

Understanding the Data: Culminating Exercise

Price per square foot of total property size



Data from Redfin