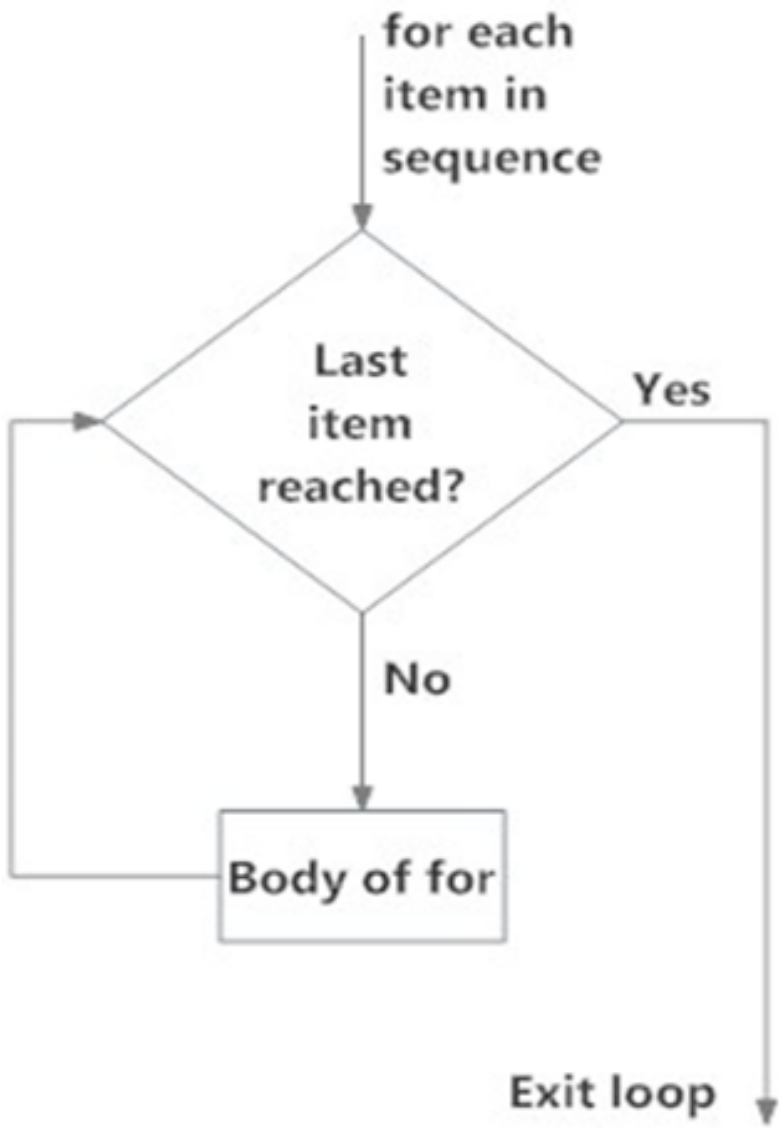
Module 2

March 9, 2018

Day 2

Recap Last Week



Simple example

- Squaring 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
```

Recap Last Week

- 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
}
```

Goals for Today

Economics - Question of the Day

- How does proximity to a metro station impacts the sale price of residential property?

What other things will affect the property value?

Goals for Today

- Determinants of property values
  - Size of house (square footage)
  - Size of property plot
  - Higher floors vs lower floors
  - Quality of local schools
  - Number of bedrooms and bathrooms
  - Neighborhood activities

Goals for Today

Programming - R

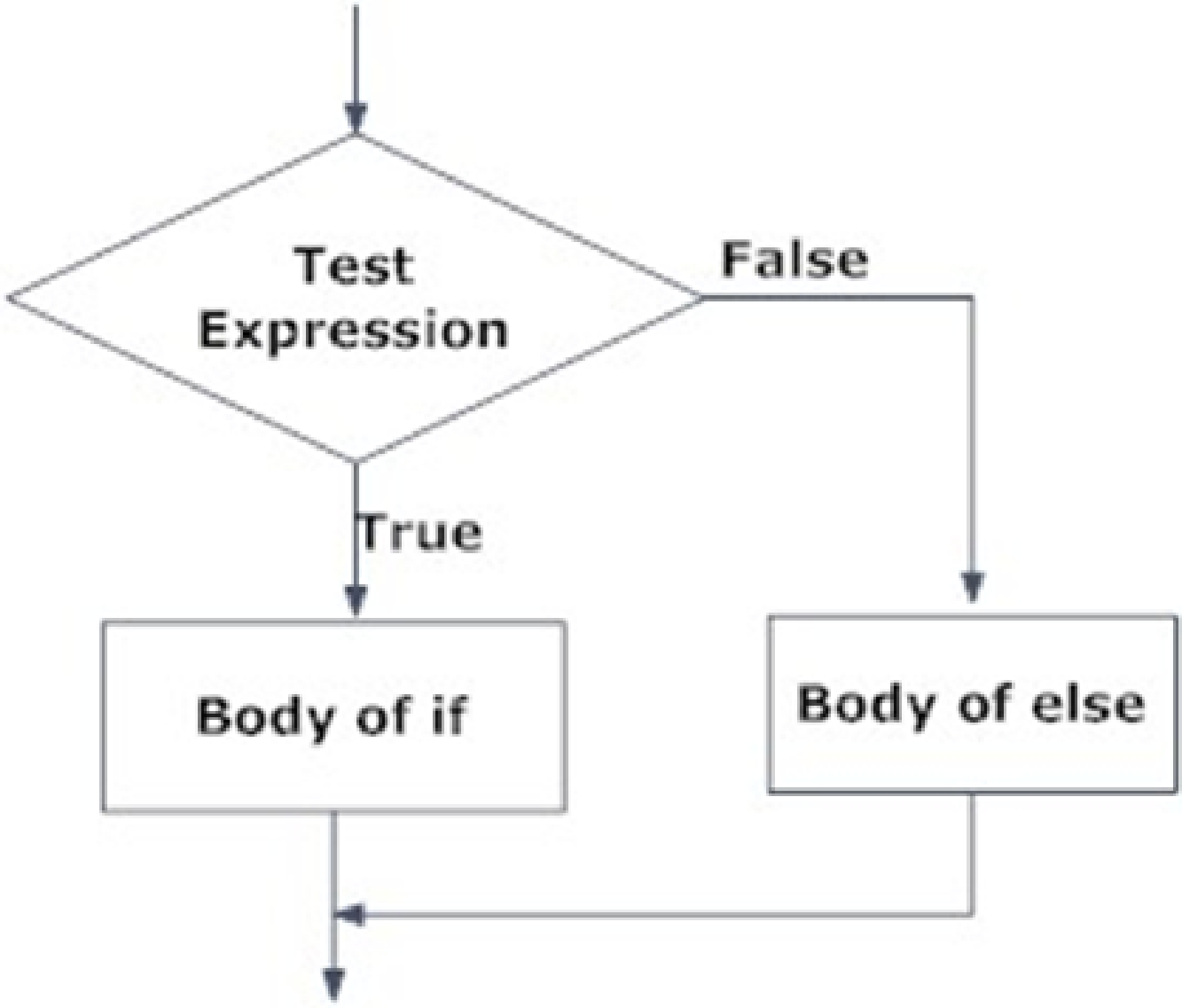
- If/Else Statements, `str_detect`, `bindrows` to append multiple datasets
- Joining multiple data frames together using dplyr `join` functions
- Cleaning data for effective data visualization

If Else Statements

- We have previously used `ifelse()` to make decisions about recoding our variables with `mutate`
- If/else statements are 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



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
```

Evens and Odds

- 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 and Odds

```
evens
```

```
## [1]  2  4  6  8 10
```

```
odds
```

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

In-Class Exercise

Given the following grade, use an if else statement to determine if the student passed or failed (cutoff is 60, with a 60 being a pass)

```
student_grade <- 71
```

Return a correct statement that the student passed or failed

If else 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 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

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")
    }
}
letter_grades
```

Assigning Grades

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

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 datasets we need a list of all of the datasets
- 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 dataset names
property_files <- paste0("Data/property_redfin_0",
    seq(1:8), ".csv")
location_files <- paste0("Data/location_redfin_0",
    c(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 datasets, but we still want our loop to decide which datasets to combine together
  - Need to know whether a dataset has location or property data
- `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)
```

In-Class Exercise

- 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_
## [3] "Data/location_redfin_03.csv" "Data/location_redfin_
## [5] "Data/location_redfin_05.csv" "Data/location_redfin_
## [7] "Data/location_redfin_07.csv" "Data/location_redfin_
```

Reading in the data - In-Class Exercise

- Now we have all the tools we need to build our dataset!
- Read in the location and property data using for loops and if else statements.
  - Within the loop, combine all of the location data into one dataset, and all of the property data into one dataset. **Hint(check out what the `bind_rows` function does.)**

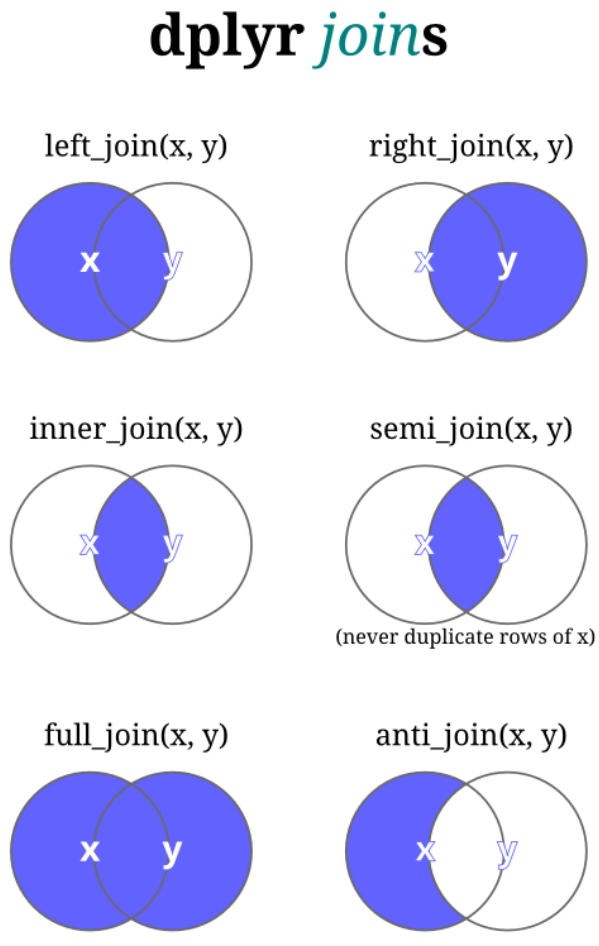
```
property_data <- data_frame()
location_data <- data_frame()

for(  ){

    if(str_detect(   )){ # You only want to have property
        data <- read_csv(   )
        property_data <- bind_rows(property_data, data)
    } else if(str_detect(file, "location")){ # Now let's c
        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
- 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

"Joining" the Data

- Combining data frames is called "joining." There are multiple types of joins as you can see in the dplyr cheat sheet:
- Help -> cheat sheets -> Data transformation with dplyr.



"Joining" the Data

- Harnessing the power of the %in% function

```
vec1 <- c(1, 5, 8, 10, 3, 7, 9)
vec2 <- c(4, 5, 19, 10, 5, 1, 8)
vec1 %in% vec2
```

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

```
which(vec1 %in% vec2)
```

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

```
# position in vec1 with overlapping elements in
# vec2
vec1[vec1 %in% vec2]
```

```
## [1]  1  5  8 10
```

```
# we can even get back those elements that
# overlap
```

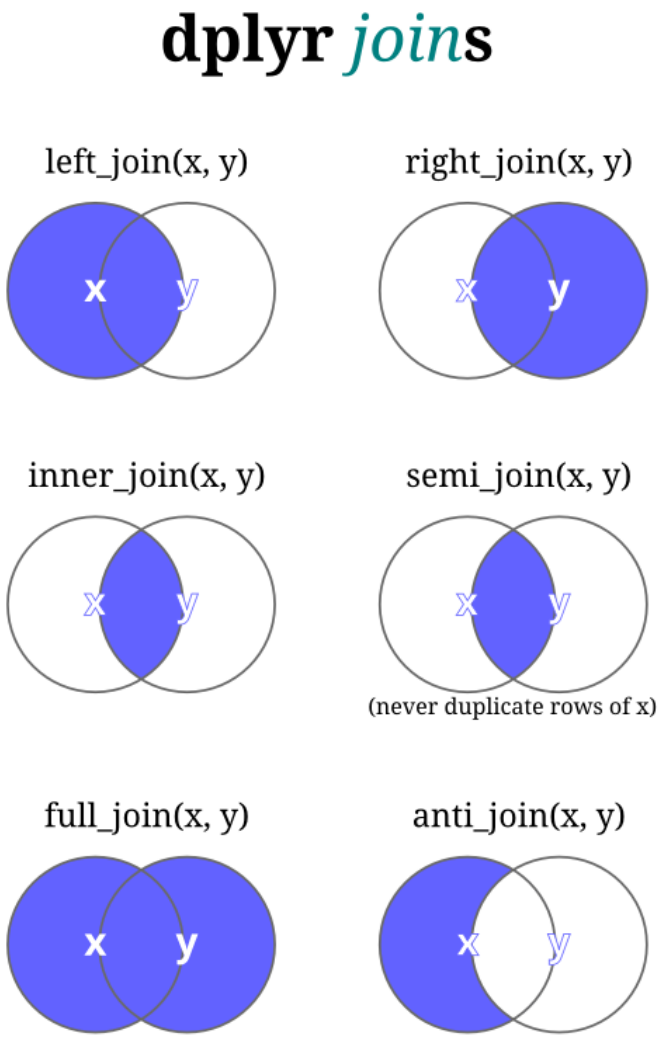
“Joining” the Data

- Now using the example we just discussed as a guide, find the overlapping variables in our property and location datasets.
- Make sure to save that overlap result into a variable so we can use it for our next step.

“Joining” the Data

- Our URL column exists in both of the dataframes!
- We only want houses that include location data...
- Which type of joining would we need to accomplish this?

“Joining” the Data



"Joining" the Data

```
joined_data <- full_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

- Success! Now all of our housing data is combined
- 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 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

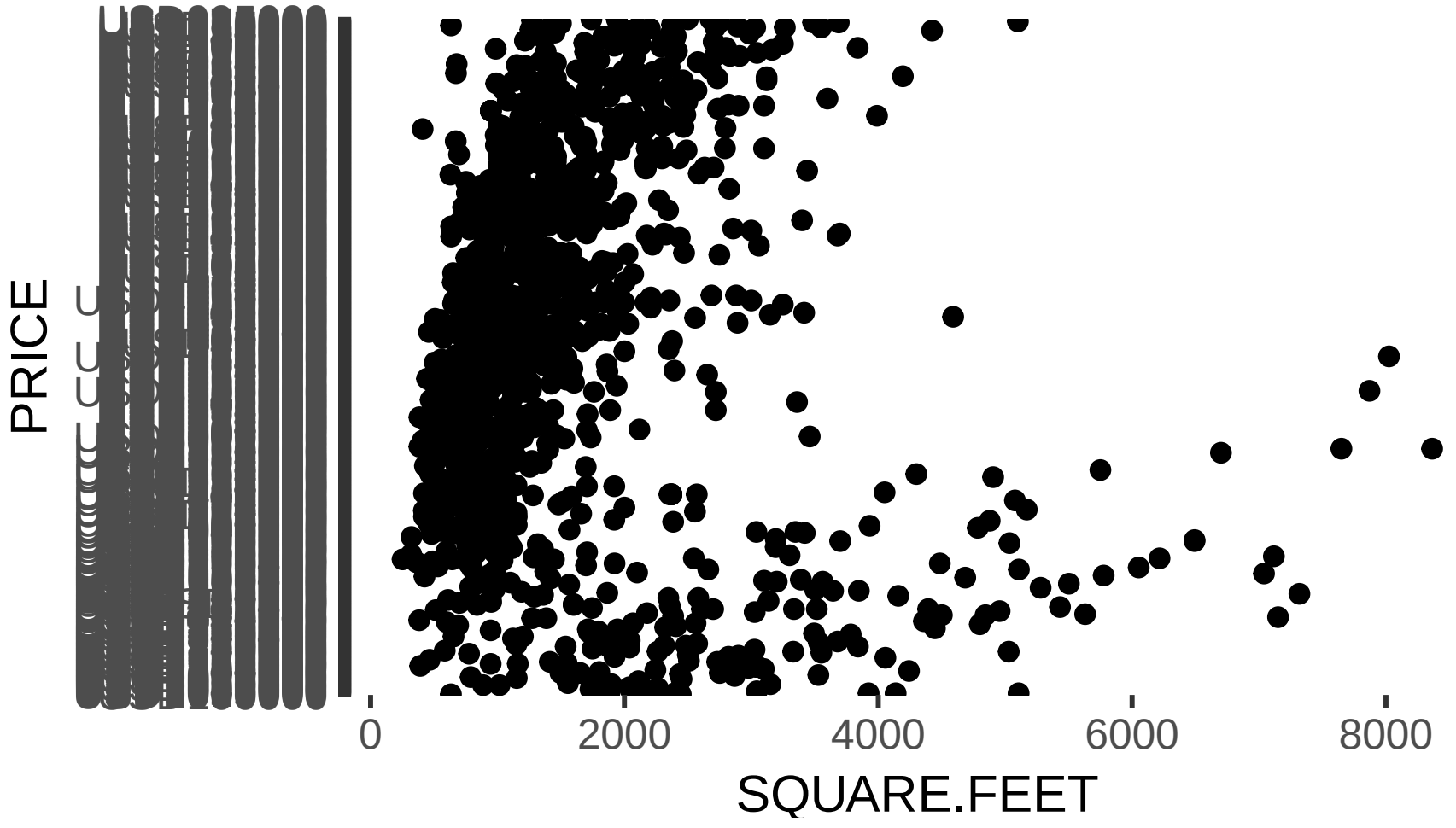
Lots of Issues Here

- city_state column - state is inconsistent (va, VA, Virginia) and unwanted states
- propertytype column - has "propertytype" in front of every observation
- zip_code column - extra zeros
- lat_lon column - & in the middle of the string
- sold.dates, list.date columns - not date objects

So let's first make a scatter-plot of our home price vs square feet variables.

Cleaning the Data

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



Cleaning the Data

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

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

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

Cleaning the Data

- How can we fix this problem for a string variable? What function could we use?

Cleaning the Data - In-Class Exercise

- One way we could do this is by replacing the USD with nothing, “”. This would have the same effect as removing it.
- Try using the str_replace function to modify the string “USD40000” to be “40000”

Cleaning the Data

- 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)
```

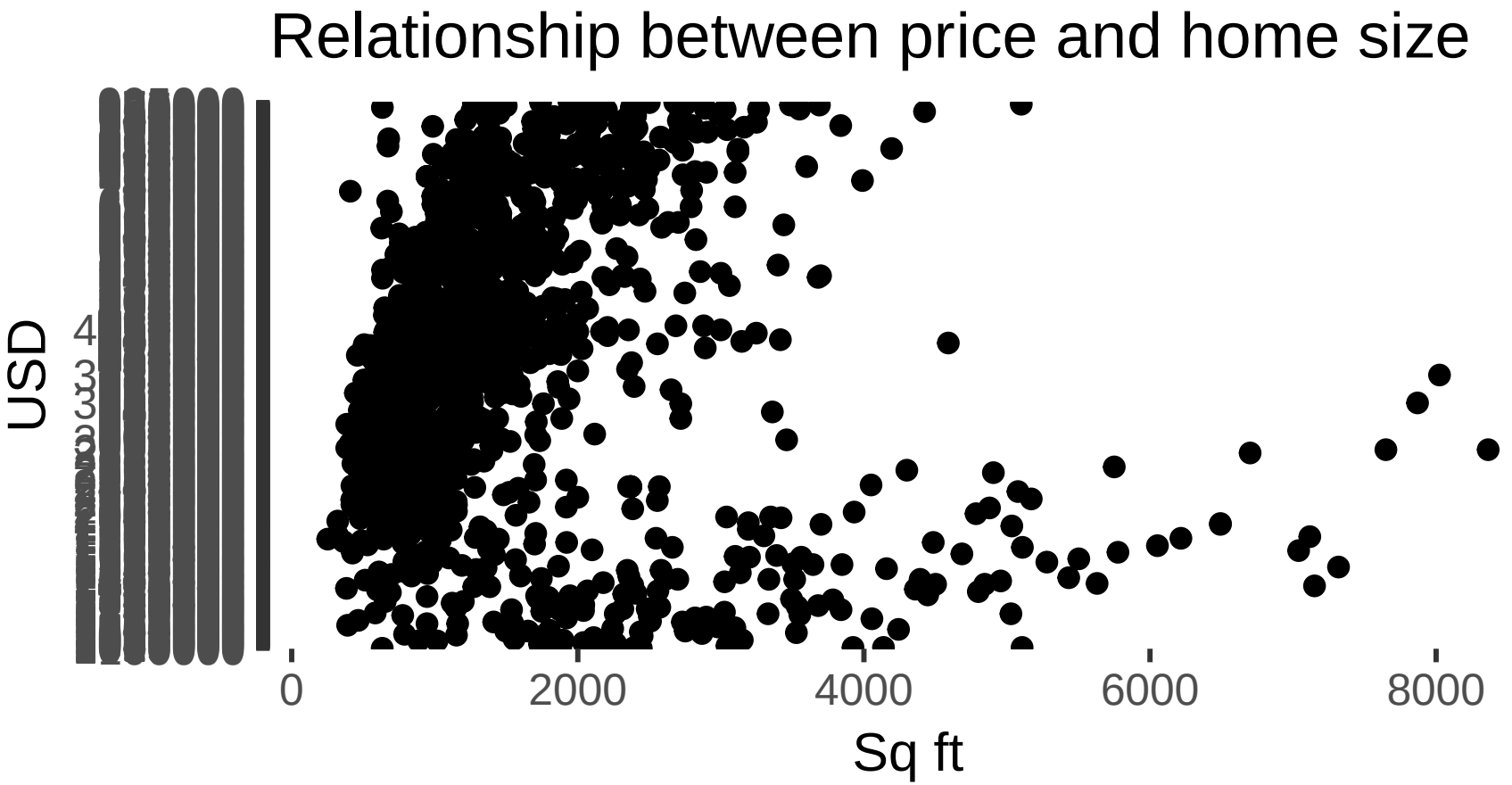
Cleaning the Data - In-Class Exercise

- We just used str_replace to fix our PRICE column.
- Check out the PROPERTY.TYPE column and use `str_replace` to fix it

Cleaning the Data

- Let's redo the plot!

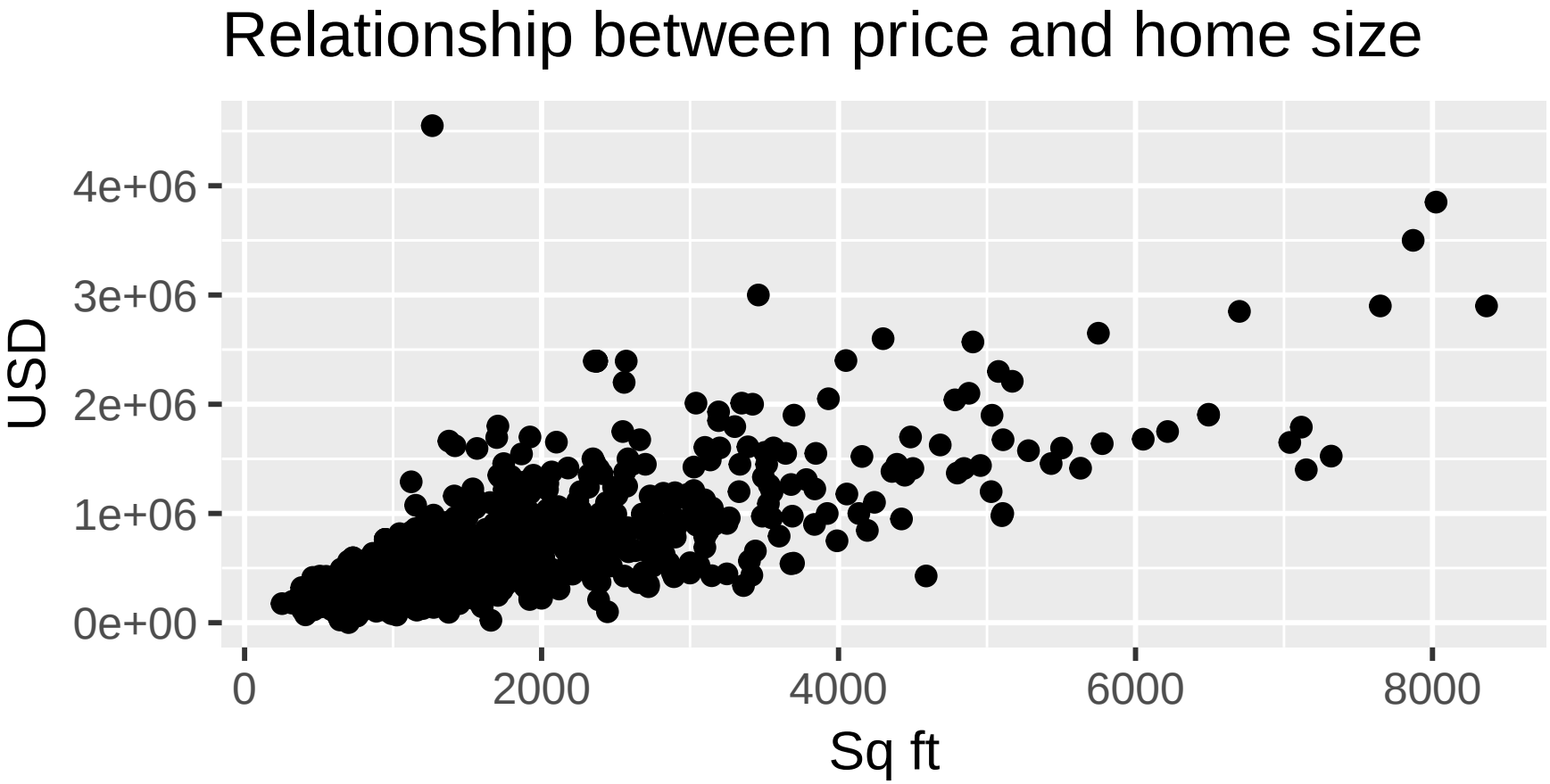
```
## Warning: Removed 13 rows containing missing values (geom
```



Cleaning the Data - In-Class Exercise

- We still have the same problem, we forgot to convert our PRICE column to be class numeric!
- 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



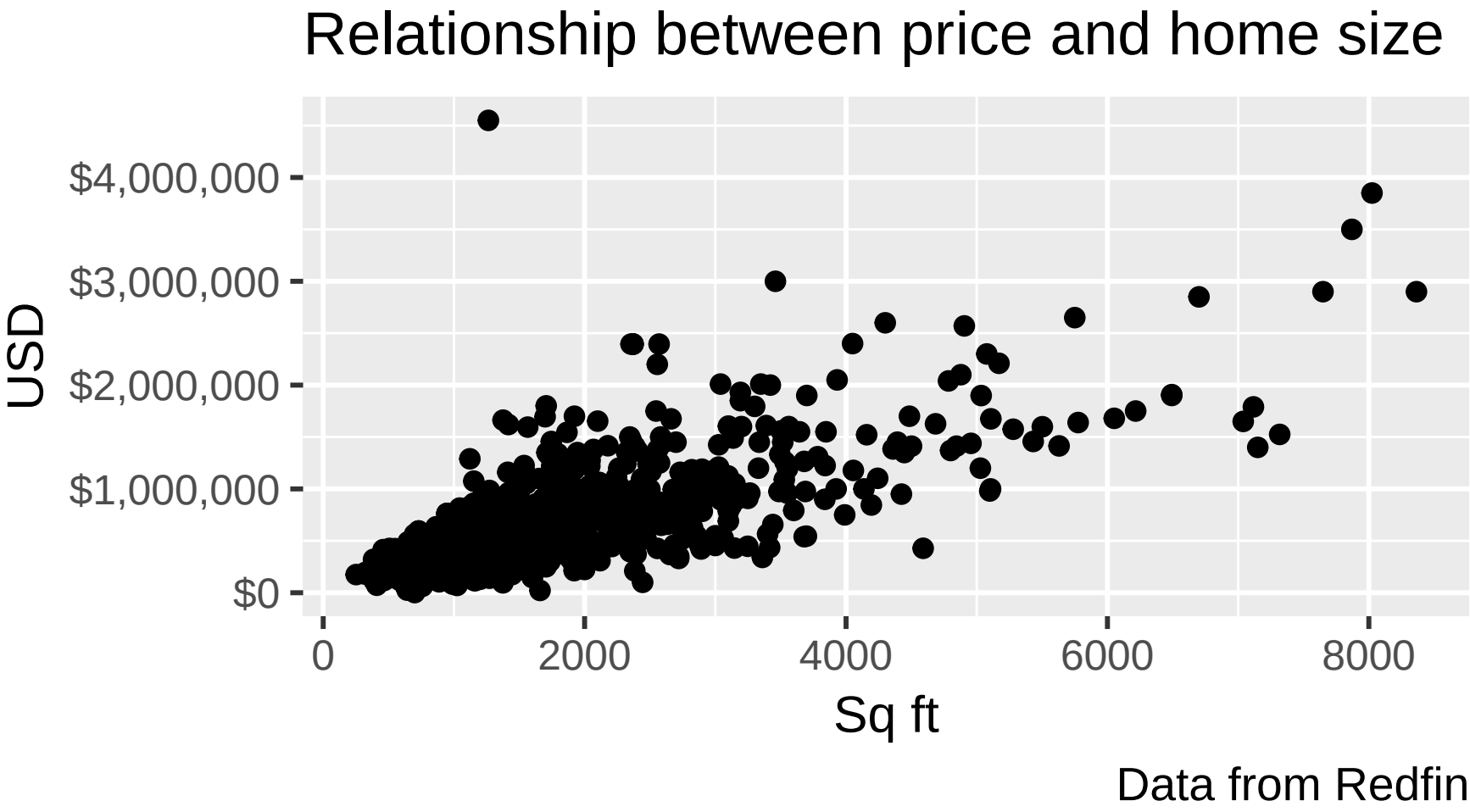
- I want to change the scale so we don't have to deal with exponentials

Cleaning the Data

- By manually entering our scale for the y-axis, we can fix the problem with exponentials on the y-axis
- `scale_y_continuous("USD", labels = dollar)`

Cleaning the Data

- What relationship is there between price and home size?



Understanding the Data

- What about property type and price?
- Is there a relationship between these two variables?
- Produce a table of the average price per square foot by property type and make a bar plot of it.

Understanding the Data

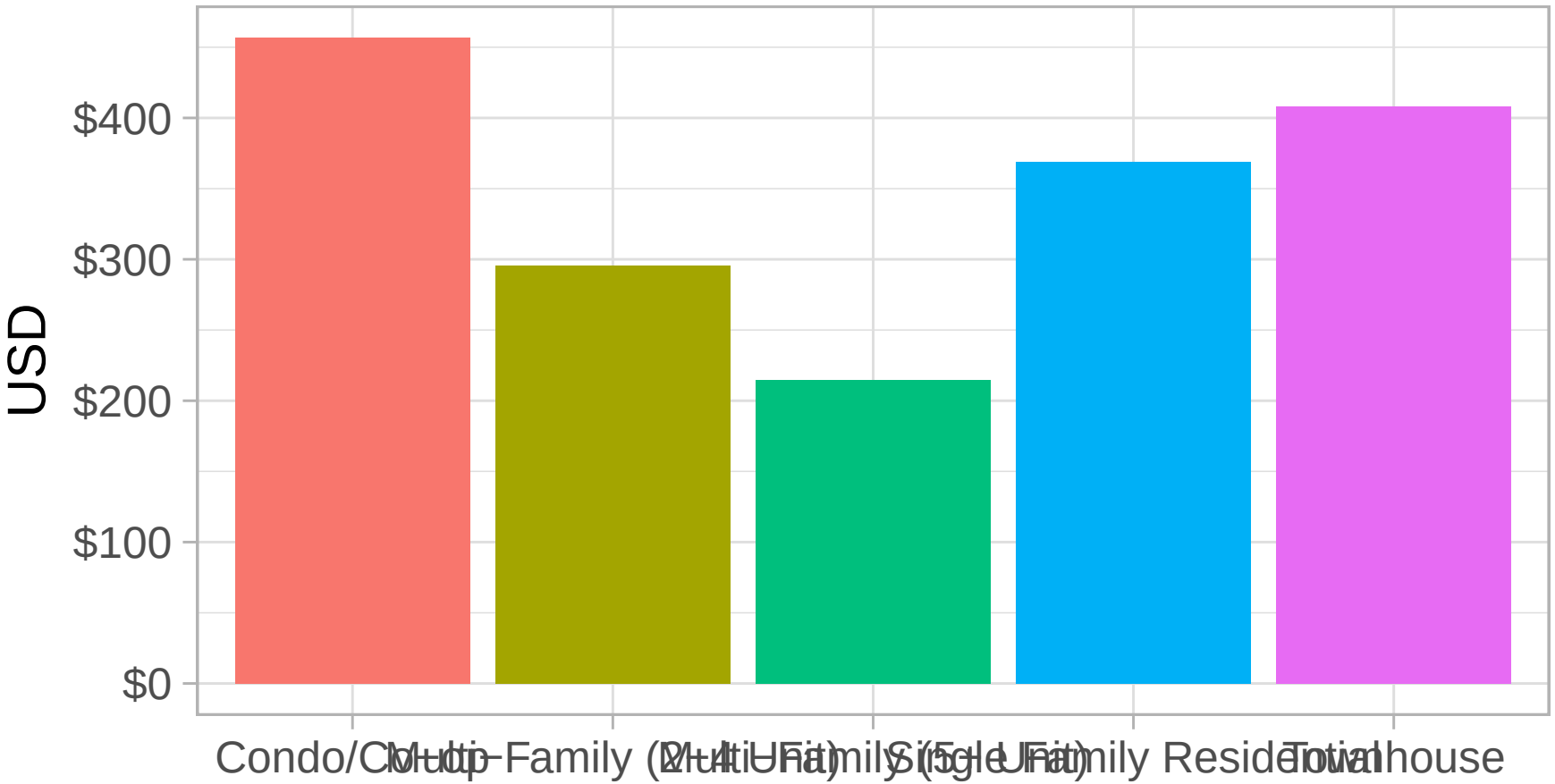
```
type_price <- joined_data %>% group_by(PROPERTY.TYPE) %>%
    summarise(price = mean(PRICE/SQUARE.FEET, na.rm = T))

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

Luckily this is not a production-quality graph, just something for our reference.

- How does theme_light change the graph? How is color assigned?



Understanding the Data

- 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

- 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

```
strings <- str_split_fixed("United States of America",
    "of", n = 2)

strings
```

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

Cleaning the Data

- We told R to split into two piece (n = 2), so we got two columns in return.
- If we want to get back to individual elements, we can subset the variable.

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

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

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

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

Cleaning the Data

- Let's test it out with something closer to our actual use case.

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

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

Cleaning the Data - In-Class Exercise

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

Cleaning the Data

```
## # 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 - In-Class Exercise

- We also need to clean the LAT_LON column by splitting it into latitude and longitude columns
- Right now our column is separated by & character, so let's split it using the method we just used

Cleaning the Data

- We have one more problem - there are spaces around our new state variable

```
head(joined_data$STATE)
```

```
## [1] " va" " va" " va" " va" " va" " va"
```

- The function `str_trim` can be used to fix this

Cleaning our Data

- All the blank spaces are removed!

```
str_trim(" word ")  # It removes the blank space!
```

```
## [1] "word"
```

- Now let's do it for our data

Cleaning the Data

- Voila! All blank spaces gone.

```
joined_data <- joined_data %>% mutate(STATE = str_trim(STATE))

head(joined_data$STATE)
```

```
## [1] "va" "va" "va" "va" "va" "va"
```

Cleaning the 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 - In-Class Exercise

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

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

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

- Now use this function to mutate the state column to upper case

Cleaning the Data

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

unique(joined_data$STATE)
```

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

Cleaning the Data - In-Class Exercise

- Now we need to change the instances of “VIRGINIA” to “VA”
- We’ve already used this function!
- Go back and find the function you need to make that change

Cleaning the Data - In-Class Exercise

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

Cleaning the Data

```
joined_data <- joined_data %>% filter(STATE %in%
    c("VA", "DC", "MD"))

unique(joined_data$STATE)
```

```
## [1] "VA" "MD" "DC"
```

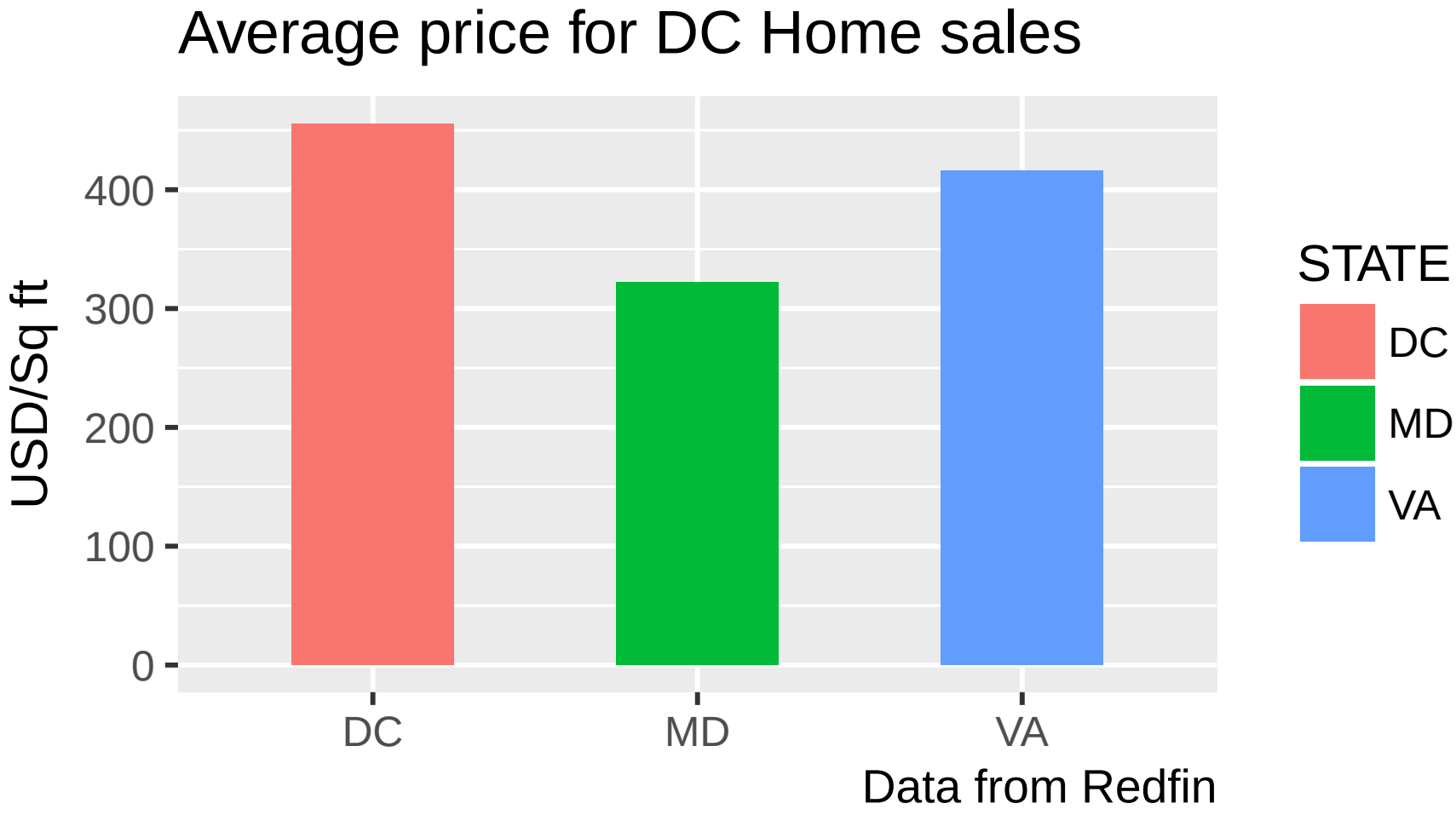
Plotting the Data

- 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

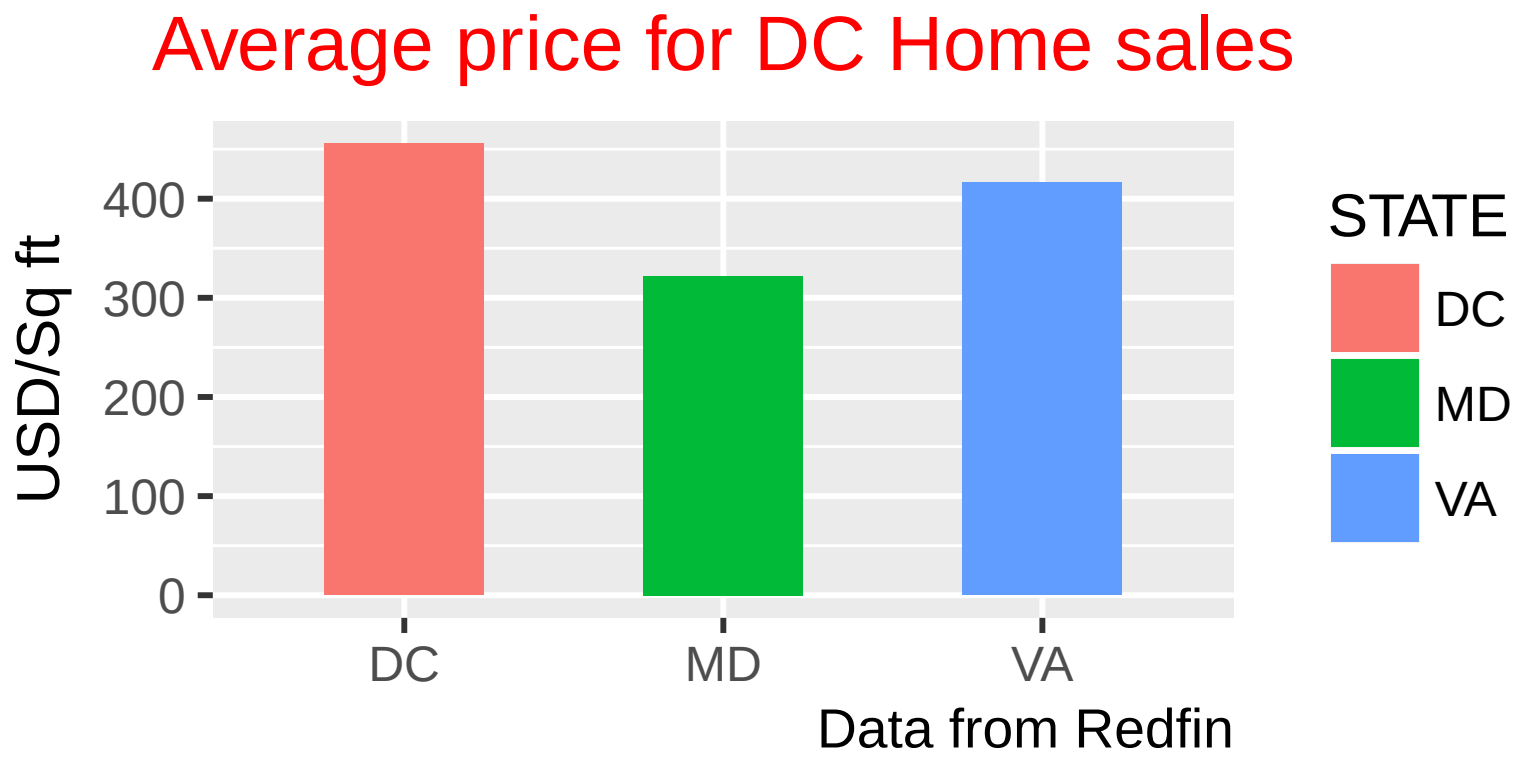
- I want to have a larger plot title and have it centered
- How can we do that? Would we use a data-related dimension or an aesthetic option?



Plotting the Data

- 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"))
```



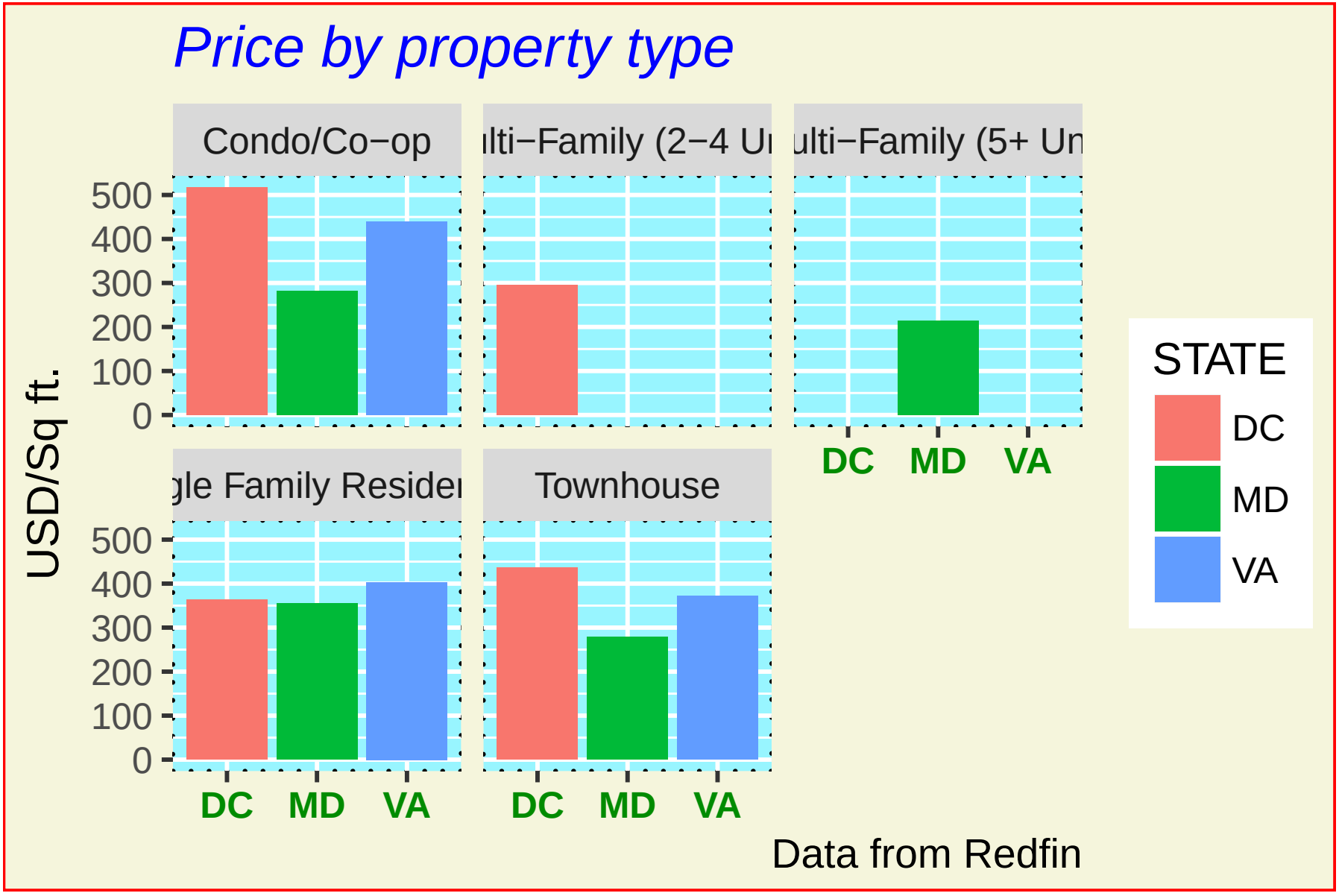
Plotting the Data

- Since our title, and the other labels, is a character string, or text, we use element_text to control it. 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)
  - vertical adjustment
  - angle
  - other aspects that impact the display of text.

Plotting the Data

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

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"), plot.background = element_rect(fill = "beige",
    color = "red"), panel.background = element_rect(fill = "cadetblue1",
    color = "black", linetype = 3))
```

Plotting the Data



Understanding the Data

- 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?
- What other rectangles could we adjust?

Understanding the Data

- 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

- 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 - In-Class Exercise

- Let's mutate the lot.size column so that if there is an `NA` then we replace it with the value of the square.feet column and otherwise keep lot.size the same

Understanding the Data

```
joined_data <- joined_data %>% mutate(LOT.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/LOT.SIZE,
    na.rm = T))

head(lot_state_data)
```

Understanding the 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

- The culminating exercise of the day!
- Make a plot like the one we just did but using our new price/lot.size variable with some modifications -
  - **Make your title centered and Green**
  - **Make your panel background White with a red border**

Understanding the Data