Midterm I Study Guide and Review and Solutions

Bastola

April 23 2024

Exam I Study Guide

Format: In Class with open-ended questions.

One-sided Cheat-sheet allowed (A4 paper)

Official `ggplot`, `dplyr`, `tidyr`, `lubridate`, `forcats`, `stringr` cheat-sheet will be provided for reference, if needed. You do not need to bring these with you.

- You are not permitted to use a laptop or classroom computer.

Topics

- The exam covers the following topics introduced in class (through Mon. 04/22):
  1. *Data types.* Understand the basic data types in R, including how to access elements in a list.
  2. *Visualizing Data.* Be able to use principles of visual perception to identify and apply appropriate visual cues in a graphic, and to read and write code to create visualizations of data using `ggplot2`.
  3. *Single and two-table dplyr verbs.* Understand the logic and implementation of the essential data wrangling tasks discussed in class.
  4. *Tidy data principles.* Understand the implementation of tidy data principles including appropriate data import conventions, and long to wide transformations and vice-versa.
  5. *Working with Dates and Times using lubridate.* Basic familiarity with `lubridate` functions for manipulating date-time objects in R.
  6. *Categorical Data Manipulation using forcats.* Understand the basics of handling categorical data using `forcats`, including factors reordering and summarizing.
  7. *String Manipulation using stringr*. Gain proficiency in handling and analyzing text data with stringr, focusing on functions for detecting patterns, performing replacements, and extracting meaningful information from strings, essential for effective text data processing.
- You will be tested on your understanding of the R code we have discussed. I will not make you write extremely complicated code from scratch, but be prepared to write small chunks of code. Additional ways I could assess your understanding of R include (but are not limited to):
  - Filling in missing arguments/lines of code.
  - Identifying the error in written code.
  - Putting lines of code in order to complete a specified task.
  - Describing the output resulting from a code chunk, including dimensions.

Practice Problems

1. An introduction to the data

Our data set contains information from the Ames Assessor's Office used in computing assessed values for individual residential properties sold in Ames, IA, from 2006 to 2010. The data contains 82 variables measured on each of 2930 properties, but we will focus on the following 12 variables:

- neighborhood = one of 28 neighborhoods in Ames
- price = sale price in $
- yrbuilt = year of original construction
- yrsold = year house was sold (2006 – 2010)
- sqft = total interior square footage
- garage = size of garage in square feet
- cars = number of cars garage holds
- lotarea = lot size in square feet
- contour = flatness of property (Lvl = level, Bnk = banked, HLS = hillside, Low = depression)
- condition = overall condition of house (10 = very excellent to 1 = very poor)
- kitchen = kitchen quality (Ex = excellent, Gd = good, TA = typical/average, Fa = fair, Po = poor)
- centralair = central air conditioning? (Y or N)

Warning: this solution is very succinct and is intended as a guide. These are not thorough answers to the problems.

What does the following code do?

Provide a thorough and intuitive (2-3 sentences) description of the output from each of the following R chunks. The chunks either produce a new data set or a new plot; if it's a new data set, give the dimensions in addition to your description. Write your descriptions in regular English, without using variable names.

a)

```
ames %>%
  group_by(neighborhood) %>%
  summarise(count = n(),
            medprice = median(price, na.rm = TRUE),
            IQRprice = IQR(price, na.rm = TRUE)
            ) %>%
  arrange(desc(medprice))
```

```
## # A tibble: 28 x 4
##    neighborhood count medprice IQRprice
##    <chr>        <int>    <dbl>    <dbl>
##  1 StoneBr         51  319000   168216
##  2 NridgHt        166  317750   125280.
##  3 NoRidge         71  302000    78605
##  4 GrnHill          2  280000    50000
##  5 Veenker         24  250250   100250
##  6 Timber          72  232106.  102494.
##  7 Somerst        182  225500    74375
##  8 Crawfor        103  200624    83550
##  9 CollgCr        267  200000    67375
## 10 Greens           8  198000    28562.
## # i 18 more rows
```

Answer:

The code groups the properties by neighborhood and calculates the total number, the median price and the interquartile range of the price for the houses for each neighborhood. Further, the code arranges the

neighborhood in descending order of the median prices. The dimension of the output is 28 × 4.

b)

```
ames %>%
  group_by(neighborhood) %>%
  summarise(newvar = mean(contour == "Lvl")) %>%
  top_n(-3, newvar)
## # A tibble: 3 x 2
##   neighborhood newvar
##   <chr>         <dbl>
## 1 ClearCr       0.432
## 2 Crawfor       0.534
## 3 StoneBr       0.529
```

Answer:

The code groups the properties by neighborhood and calculates the proportion of the level contour properties in each neighborhood. The code then outputs the neighborhoods with three of the lowest proportions of level contour houses. The dimension of the output is 3 × 2.

c)

```
ames %>%
  group_by(neighborhood) %>%
  filter(n() > 2) %>%
  top_n(3, price) %>%
  select(neighborhood, price, yrbuilt, sqft, garage, condition) %>%
  arrange(neighborhood, desc(price))
## # A tibble: 81 x 6
## # Groups:   neighborhood [26]
##    neighborhood  price yrbuilt  sqft garage condition
##    <chr>         <dbl>   <dbl> <dbl>  <dbl>     <dbl>
##  1 Blmngtn      264561    2006  3087    648         5
##  2 Blmngtn      246990    2002  2879    398         5
##  3 Blmngtn      246578    2006  2856    648         5
##  4 Blueste      200000    1988  2024    509         5
##  5 Blueste      185000    1989  2016    598         6
##  6 Blueste      162500    1989  2382    531         5
##  7 BrDale       125500    1973  2130    440         6
##  8 BrDale       125000    1970  2130    440         6
##  9 BrDale       122500    1971  2130    440         5
## 10 BrkSide      223500    1923  2900    528         6
## # i 71 more rows
```

Answer:

The code groups the properties by neighborhood and filters the neighborhood that are appear more than 2 times in the dataset, and selects the top 3 instances in terms of price for each neighborhood. The code then selects the said variables and then arranges them in descending order by price. The dimension of the output is 81 × 6. (I believe now that the dimension is impossible to know here. Sorry about that! There is one neighborhood that appears 2 times or less.)

2. Joining Data

Based on the given tribbles, `df_primary` and `df_secondary`, answer the following questions.

```
library(dplyr)
df_primary <- tribble(
  ~ID, ~y,
   "A", 5,
   "B", 5,
   "C", 8,
   "D", 0,
  "F", 9)
df_secondary <- tribble(
  ~ID, ~z,
   "A", 30,
   "B", 21,
   "C", 22,
   "D", 25,
   "E", 29)
```

a) What would be the output of the following function?

```
full_join(df_primary, df_secondary, by ='ID')
## # A tibble: 6 x 3
##   ID        y     z
##   <chr> <dbl> <dbl>
## 1 A         5    30
## 2 B         5    21
## 3 C         8    22
## 4 D         0    25
## 5 F         9    NA
## 6 E        NA    29
```

b) What would be the output of the following function?

```
right_join(df_primary, df_secondary, by ='ID')
## # A tibble: 5 x 3
##   ID        y     z
##   <chr> <dbl> <dbl>
## 1 A         5    30
## 2 B         5    21
## 3 C         8    22
## 4 D         0    25
## 5 E        NA    29
```

c) What would be the output of the following function?

```
anti_join(df_primary, df_secondary, by ='ID')
## # A tibble: 1 x 2
##   ID        y
##   <chr> <dbl>
## 1 F         9
```

3. Cleaning Messy Data

```
# Create a messy dataset
messy <- data.frame(
```

```
  country = c("A", "B", "C"),
  q1_2021 = c(0.03, 0.05, 0.01),
  q2_2021 = c(0.05, 0.07, 0.02),
  q3_2021 = c(0.04, 0.05, 0.01),
  q4_2021 = c(0.03, 0.02, 0.04))
messy
##   country q1_2021 q2_2021 q3_2021 q4_2021
## 1       A    0.03    0.05    0.04    0.03
## 2       B    0.05    0.07    0.05    0.02
## 3       C    0.01    0.02    0.01    0.04
```

Following are the steps in tidying up the above data so that the data follows tidy principles. Briefly state the steps you would follow in making the above data tidy.

a) Now fill in the blanks to achieve your tidy data goals. What would the dimension of the resulting data tibble be?

```
tidier <- messy %>%
  pivot_longer(cols = #### FILL IN 1 ###,
               names_to = "quarter",
               values_to = "growth")
```

```
tidier <- messy %>%
  pivot_longer(cols = q1_2021:q4_2021,
               names_to = "quarter",
               values_to = "growth")
tidier
## # A tibble: 12 x 3
##    country quarter growth
##    <chr>   <chr>    <dbl>
##  1 A       q1_2021   0.03
##  2 A       q2_2021   0.05
##  3 A       q3_2021   0.04
##  4 A       q4_2021   0.03
##  5 B       q1_2021   0.05
##  6 B       q2_2021   0.07
##  7 B       q3_2021   0.05
##  8 B       q4_2021   0.02
##  9 C       q1_2021   0.01
## 10 C       q2_2021   0.02
## 11 C       q3_2021   0.01
## 12 C       q4_2021   0.04
```

b) How would you separate quarter column in tidier tibble to make your data the tidiest? Please write your code below.

```
tidiest <- tidier %>% separate(quarter, c("quarter", "year"))
tidiest
## # A tibble: 12 x 4
##    country quarter year  growth
##    <chr>   <chr>   <chr>  <dbl>
##  1 A       q1      2021    0.03
##  2 A       q2      2021    0.05
```

```
##  3 A       q3      2021    0.04
##  4 A       q4      2021    0.03
##  5 B       q1      2021    0.05
##  6 B       q2      2021    0.07
##  7 B       q3      2021    0.05
##  8 B       q4      2021    0.02
##  9 C       q1      2021    0.01
## 10 C       q2      2021    0.02
## 11 C       q3      2021    0.01
## 12 C       q4      2021    0.04
```

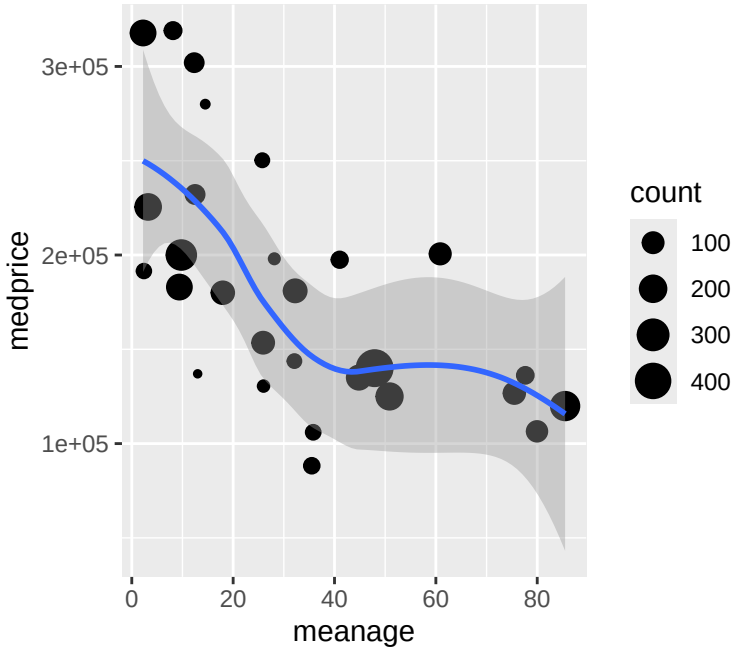
4. Fill in R code.

Provide the necessary lines of R code to produce the given data set or plot, or to complete the described task.

a) Produce the plot below. Note: `age` refers to how old the house when when it was sold.

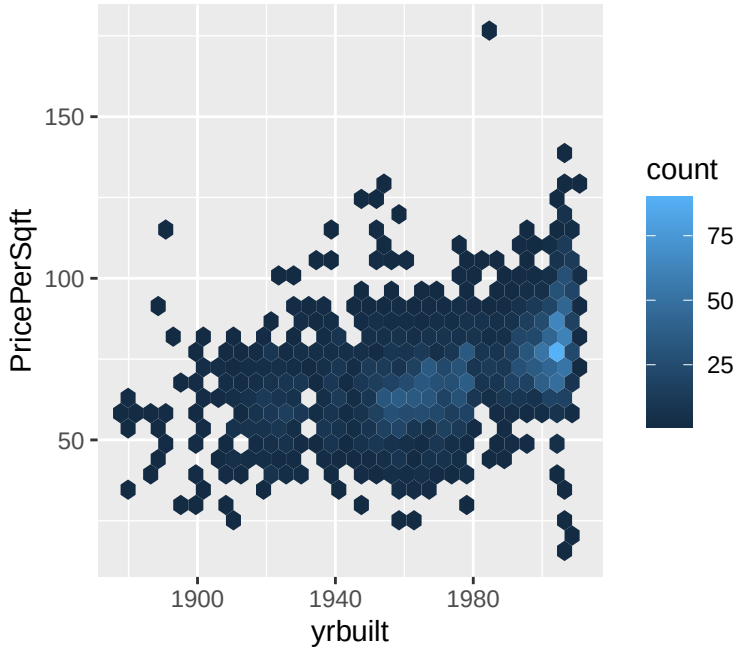
```
by_neighborhood <- ames %>%
  group_by(neighborhood) %>%
  ## LINE 1 ##
  summarise(count = n(),
            medprice = median(price, na.rm = TRUE),
            ## LINE 2 ##
            )

ggplot(data = by_neighborhood, mapping = aes(x = meanage, y = medprice)) +
  ## LINE 3 ##
  geom_smooth()
```

```
by_neighborhood <- ames %>%
  group_by(neighborhood) %>%
  mutate(age = yrsold - yrbuilt) %>%
  summarise(count = n(),
            medprice = median(price, na.rm = TRUE),
            meanage = mean(age, na.rm = TRUE)
            )

ggplot(data = by_neighborhood, mapping = aes(x = meanage, y = medprice)) +
  geom_point(mapping = aes(size = count)) +
  geom_smooth()
```

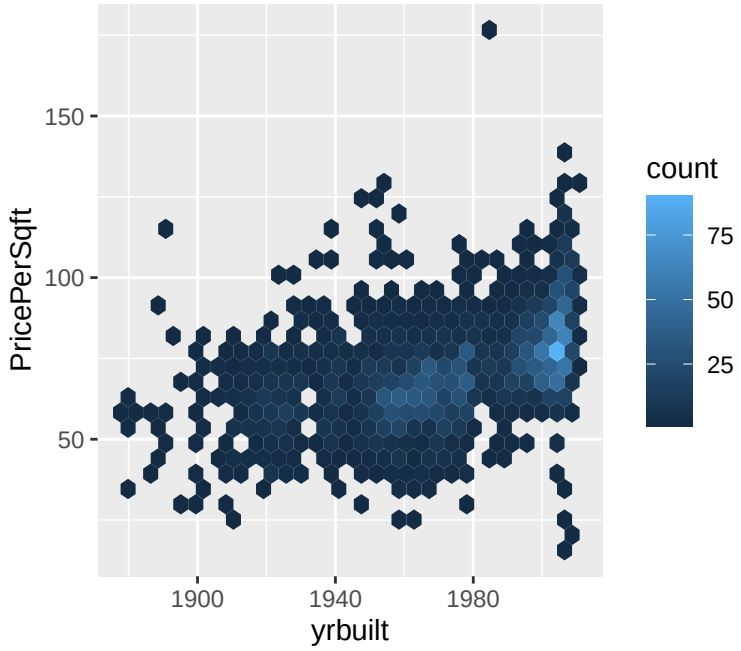


b) Rewrite the R code below to be one string of commands connected by pipes.

```
onlyair <- filter(ames, centralair == "Y")
onlyair_small <- select(onlyair, price, sqft, yrbuilt)
onlyair_small <- mutate(onlyair_small, PricePerSqft = price / sqft)
ggplot(data = onlyair_small, mapping = aes(x = yrbuilt, y = PricePerSqft)) +
  geom_hex()
```



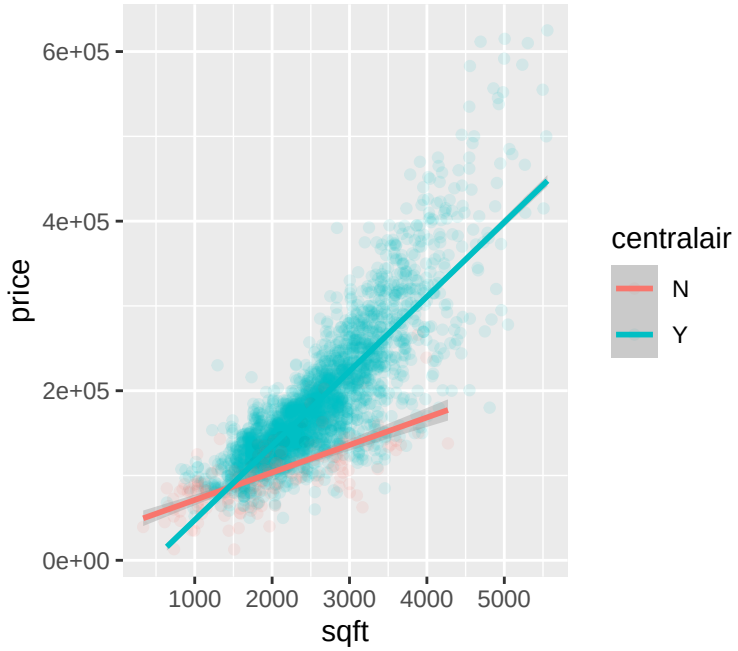

```
# Rewrite

ames %>%
  select(centralair, price, sqft, yrbuilt) %>%
  filter(centralair == "Y") %>%
  mutate(PricePerSqft = price / sqft) %>%
  ggplot(mapping = aes(x = yrbuilt, y = PricePerSqft)) +
  geom_hex()
```



5. Explain what would happen (usually 1 sentence will do) if the described changes are made in the R chunks below. Evaluate each change separately.

I)

```
ames %>%
  filter(!is.na(sqft), sqft < 6000) %>%
  ggplot(mapping = aes(x = sqft, y = price, colour = centralair)) +
    geom_point(alpha = 1/10) +
    geom_smooth(method = "lm")
```

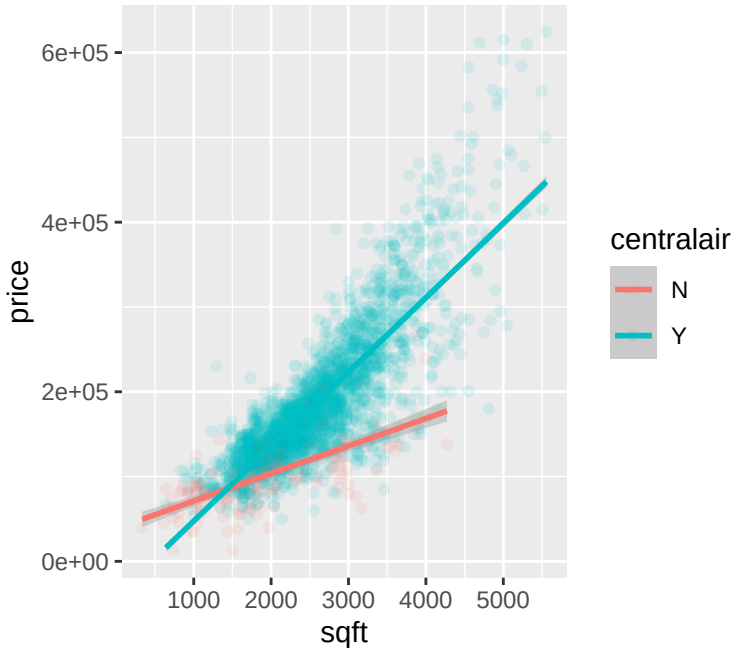


a) remove the "!" from `!is.na(sqft)`

```
ames %>%
  filter(is.na(sqft), sqft < 6000) %>%
  ggplot(mapping = aes(x = sqft, y = price, colour = centralair)) +
    geom_point(alpha = 1/10) +
    geom_smooth(method = "lm")
```

b) replace the "," after `!is.na(sqft)` with "&"

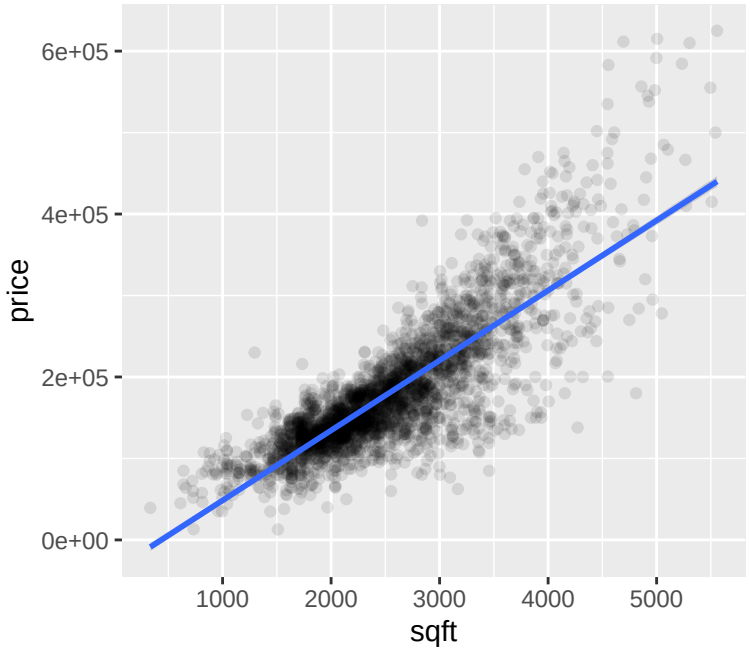
```
ames %>%
  filter(!is.na(sqft) & sqft < 6000) %>%
  ggplot(mapping = aes(x = sqft, y = price, colour = centralair)) +
    geom_point(alpha = 1/10) +
    geom_smooth(method = "lm")
```



c) replace the third line with `ggplot(mapping = aes(x = sqft, y = price), colour = centralair)`

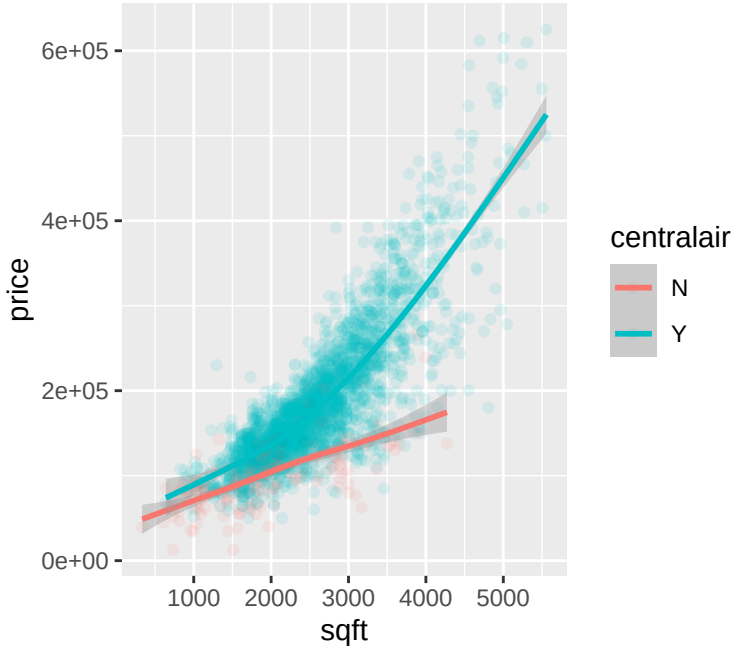
```
ames %>%
  filter(!is.na(sqft) & sqft < 6000) %>%
```

```
  ggplot(mapping = aes(x = sqft, y = price), colour = centralair) +
    geom_point(alpha = 1/10) +
    geom_smooth(method = "lm")
```



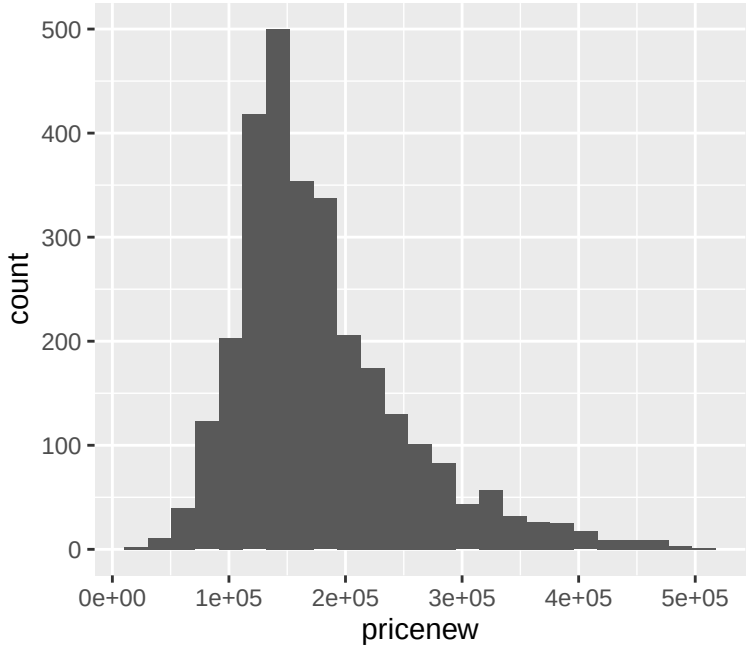
d) replace the last line with `geom_smooth()`

```
ames %>%
  filter(!is.na(sqft), sqft < 6000) %>%
  ggplot(mapping = aes(x = sqft, y = price, colour = centralair)) +
    geom_point(alpha = 1/10) +
    geom_smooth()
```



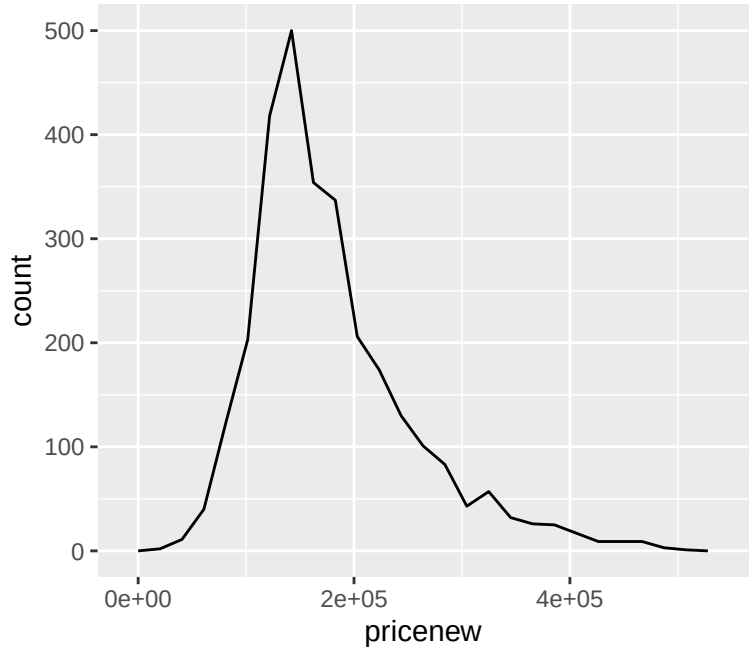
II.

```
ames %>%
  mutate(pricenew = ifelse(price > 500000, NA, price)) %>%
  ggplot(mapping = aes(x = pricenew)) +
    geom_histogram(bins = 25)
```



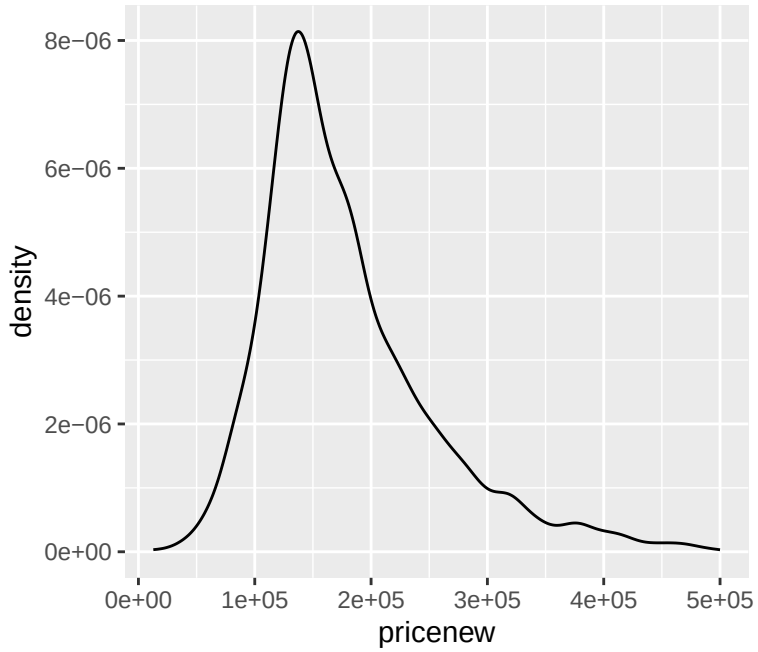
a) replace the last line with `geom_freqpoly(bins = 25)`

```
ames %>%
  mutate(pricenew = ifelse(price > 500000, NA, price)) %>%
  ggplot(mapping = aes(x = pricenew)) +
    geom_freqpoly(bins = 25)
```



b) replace the last line with `geom_density()`

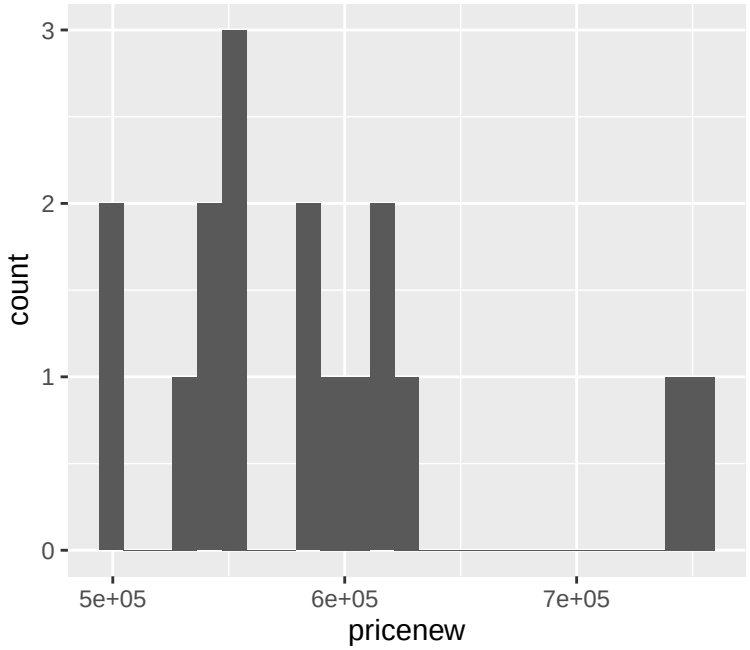
```
ames %>%
  mutate(pricenew = ifelse(price > 500000, NA, price)) %>%
  ggplot(mapping = aes(x = pricenew)) +
  geom_density()
```



c) replace the `ifelse()` statement with `ifelse(price > 500000, price, NA)`

```
ames %>%
  mutate(pricenew = ifelse(price > 500000, price, NA)) %>%
  ggplot(mapping = aes(x = pricenew)) +
```

```
    geom_histogram(bins = 25)
```



6. Consider the following objects to answer the questions below.

```
x <- -2:3
x
## [1] -2 -1  0  1  2  3
y <- data.frame(y1 = c(25, 16), y2 = c(TRUE, FALSE))
y
##   y1    y2
## 1 25  TRUE
## 2 16 FALSE
z <- list(z1 = x, z2 = y, z3 = c("good","luck!"))
z
## $z1
## [1] -2 -1  0  1  2  3
##
## $z2
##   y1    y2
## 1 25  TRUE
## 2 16 FALSE
##
## $z3
## [1] "good"  "luck!"
```

(a) Consider the objects x, y and z. Which are atomic vectors and which are lists?

Answer: x is an atomic vector, y and z are lists

(b) What does the following command evaluate to? Briefly explain your answer.

```
y$y1 + c(-1,1)*y$y2
## [1] 24 16
```

Answer: subtracts 1 from 25 and adds 0 to 16.

(c) Explain why the first line of code below returns the word "luck!" while the second line of code doesn't return anything (i.e. a null value).

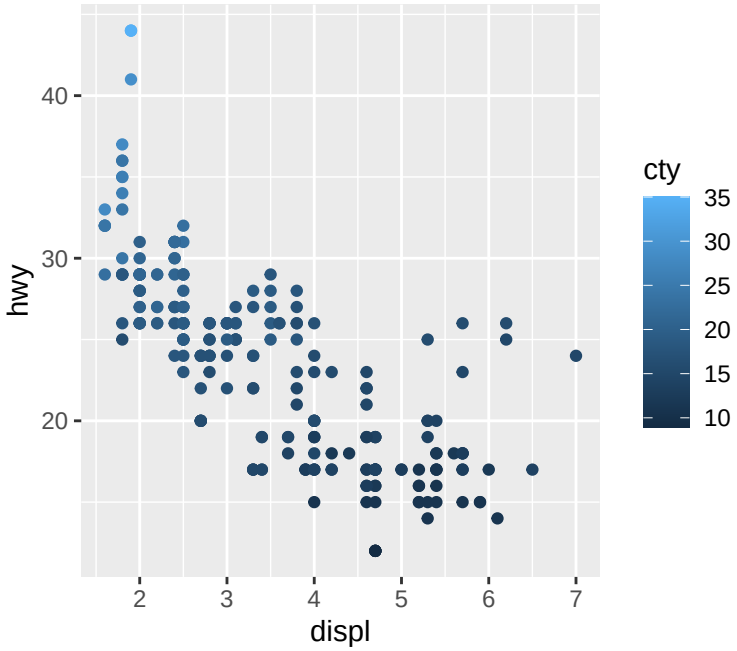
```
z[[3]][2]    # line 1
## [1] "luck!"
z[3][2]      # line 2
## $<NA>
## NULL
```

Answer: Line 1 produces "luck!" because we are extracting the second element of the vector contained in z[[3]]. Line 2 is a list of one element, so it returns NULL.

7. Miscelleaneus

a) How could you improve upon the following plot? Please provide your code modification and explain your reasoning.

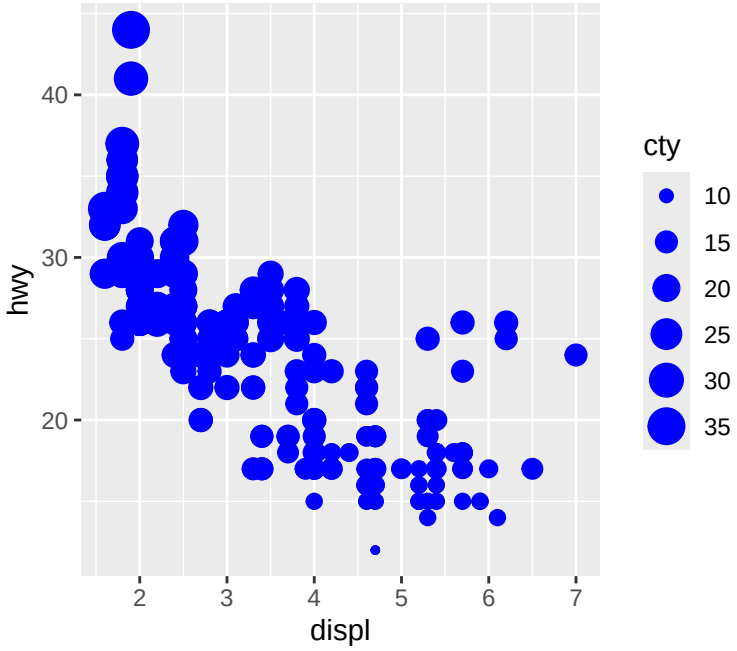
```
ggplot(mpg, aes(x = displ, y = hwy, colour = cty)) +
  geom_point()
```



Answer:

One way to improve the plot would be to use `size = city` and add a color outside of aesthetics instead.

```
ggplot(mpg, aes(x = displ, y = hwy, size = cty)) +
  geom_point(color = "blue")
```



b) What is the dimension of the resulting data tibble from the code chunk below? Explain.

```
tibble(x = sample(1:9),
       group = rep(c("a", "b", "c"), each = 3)) %>%
  mutate(x_mean = mean(x)) %>%
  group_by(group) %>%
  summarize(x_mean_2 = mean(x))
## # A tibble: 3 x 2
##   group x_mean_2
##   <chr>    <dbl>
## 1 a            6
## 2 b            5
## 3 c            4
```

Answer: The resulting tibble has 3 rows and 2 columns. The code first creates a tibble with 9 randomly sampled integers in the range 1-9 and a grouping variable group that repeats the values "a", "b", and "c" three times. It then computes the mean of x and stores it in a new variable x_mean. After that, the data is grouped by group and the mean of x is computed again, this time storing it in a new variable x_mean_2. Since there are three unique values of group, the resulting tibble has 3 rows and 2 columns.

c) What is the dimension of the resulting data tibble from the code chunk below? Explain.

```
tibble(x = runif(9),
       group = rep(c("a", "b", "c"), each = 3)) %>%
  mutate(x_mean = mean(x)) %>%
  group_by(group) %>%
  arrange(x) %>%
  slice(1)
## # A tibble: 3 x 3
## # Groups:   group [3]
##         x group x_mean
##     <dbl> <chr>  <dbl>
## 1 0.0679 a      0.625
## 2 0.138  b      0.625
## 3 0.231  c      0.625
```

Answer: The resulting tibble has 3 rows and 3 columns. The code first creates a tibble with 9 randomly generated values from a uniform distribution and a grouping variable group that repeats the values "a", "b", and "c" three times. It then computes the mean of x and stores it in a new variable x_mean. Next, the data is grouped by group, arranged in ascending order of x, and only the first row of each group is selected using

slice(1). Since there are three unique values of group, the resulting tibble has 3 rows and 3 columns.

d) What is the dimension of the resulting data tibble from the code chunk below? Explain.

```
tibble(group = rep(c("a", "b", "c"), each = 3),
       x = runif(9)) %>%
  group_by(group) %>%
  arrange(x) %>%
  mutate(lag_x = lag(x)) %>%
  tidyr::drop_na(lag_x)
## # A tibble: 6 x 3
## # Groups:   group [3]
##   group     x  lag_x
##   <chr> <dbl>  <dbl>
## 1 c     0.274 0.0445
## 2 b     0.474 0.227
## 3 a     0.478 0.278
## 4 c     0.719 0.274
## 5 b     0.726 0.474
## 6 a     0.916 0.478
```

Answer: The resulting data tibble will have 6 rows and 3 columns. The code first creates a tibble with 9 rows and 2 columns, where each group of 3 rows has the same value of "group". It then groups the data by "group", sorts each group in ascending order by "x", calculates the lagged value of "x" within each group, drops any rows with missing values, and returns the resulting tibble. Since there are 3 unique groups and each group has 2 non-missing values of "x" (excluding the first row in each group, which has a missing lagged value), the resulting tibble will have 6 rows and 3 columns.

e) What does the following code chunk do?

```
library(forcats)
ames <- ames %>%
  mutate(neighborhood = fct_reorder(neighborhood, price, median, na.rm = TRUE))
```

Using the `ames` dataset, the code reorders the levels of neighborhood based on the median sale price in each neighborhood, ensuring that the neighborhood with the highest median sale price comes first.

f) Suppose you have a dataset with a factor variable f having levels "low", "medium", "high". Write a code snippet to reverse the order of levels in f.

```
f <- factor(c("low", "medium", "high"), levels = c("low", "medium", "high"))
f <- fct_rev(f)
levels(f)
## [1] "high"   "medium" "low"
```

g) Assuming you have a data frame project_data that contains the columns `project_start` and project_end (both in "YYYY-MM-DD" format), write a snippet to add a new column `project_length` to this data frame. `project_length` should contain the duration of each project in terms of the number of whole weeks.

```
library(lubridate)
project_data <- data.frame(
```

```
  project_id = 1:3,
  project_start = ymd(c("2020-01-01", "2020-02-15", "2020-03-20")),
  project_end = ymd(c("2020-01-22", "2020-03-10", "2020-04-25"))
)
```

```
library(lubridate)
project_data$project_length <- as.duration(interval(start = project_data$project_start,
                                                    end = project_data$project_end)) / dweeks(1)
#project_data$project_length <- floor(project_data$project_length)
project_data %>% knitr::kable()
```

project_id	project_start	project_end	project_length
1	2020-01-01	2020-01-22	3.000000
2	2020-02-15	2020-03-10	3.428571
3	2020-03-20	2020-04-25	5.142857

8 String manipulations

a. Given a vector of words, write a function named `find_vowel_5_letter_words` that identifies all five-letter words starting and ending with a vowel. The function should return the indices of these words within the original vector. For this problem, assume vowels are "a", "e", "i", "o", and "u", and consider case insensitivity.

```
find_vowel_5_letter_words <- function(words) {
  pattern <- "^[aeiouAEIOU].{3}[aeiouAEIOU]$"
  matches <- str_detect(words, pattern)
  return(which(matches))
}

find_vowel_5_letter_words(c("achar", "irene", "utopia"))
## [1] 2
```

b. Write a command to remove all instances of # from the `comments` vector.

```
comments <- c("Loving the new #season!", "Can't wait for #Friday!")
```

```
clean_comments <- str_replace_all(comments, "#", "")
clean_comments
## [1] "Loving the new season!" "Can't wait for Friday!"
```

c. Write a command to insert dashes (-) into the dates vector to achieve the desired format.

```
dates <- c("01012023", "15032024")
```

```
formatted_dates <- str_c(
  str_sub(dates, 1, 2), "-",
  str_sub(dates, 3, 4), "-",
  str_sub(dates, 5, 8)
)
formatted_dates
## [1] "01-01-2023" "15-03-2024"
```

d. (10 points)

```
info <- "John Doe:john.doe@example.com, Jane Smith:jane.smith@example.com"
```

Use the `stringr` functions to separate the names and email addresses into two vectors, `names` and `emails`.

```
info <- "John Doe:john.doe@example.com, Jane Smith:jane.smith@example.com"

separated_info <- str_split(str_split(info, ",\\s*")[[1]], ":")

names <- c(separated_info[[1]][1], separated_info[[2]][1])
emails <- c(separated_info[[1]][2], separated_info[[2]][2])

names
## [1] "John Doe"   "Jane Smith"
emails
## [1] "john.doe@example.com"   "jane.smith@example.com"
```