HW2

2024-10-06

HW 2 - DSC 441

Problem 1

For this problem, you will load and perform some cleaning steps on a dataset in the provided BankData.csv, which is data about loan approvals from a bank in Japan (it has been modified from the original for our purposes in class, so use the provided version). Specifically, you will use visualization to examine the variables and normalization, binning and smoothing to change them in particular ways.

- a: Visualize the distributions of the variables in this data. You can choose bar graphs, histograms and density plots. Make appropriate choices given each type of variables and be careful when selecting parameters like the number of bins for the histograms. Note there are some numerical variables and some categorical ones. The ones labeled as a 'bool' are Boolean variables, meaning they are only true or false and are thus a special type of categorical. Checking all the distributions with visualization and summary statistics is a typical step when beginning to work with new data.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
loan = read_csv("/Users/danielkim/Downloads/BankData.csv")
```

```
## New names:
## Rows: 690 Columns: 13
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (1): approval dbl (9): ...1, cont1, cont2, cont3, cont4, cont5, cont6,
## credit.score, ages lgl (3): bool1, bool2, bool3
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

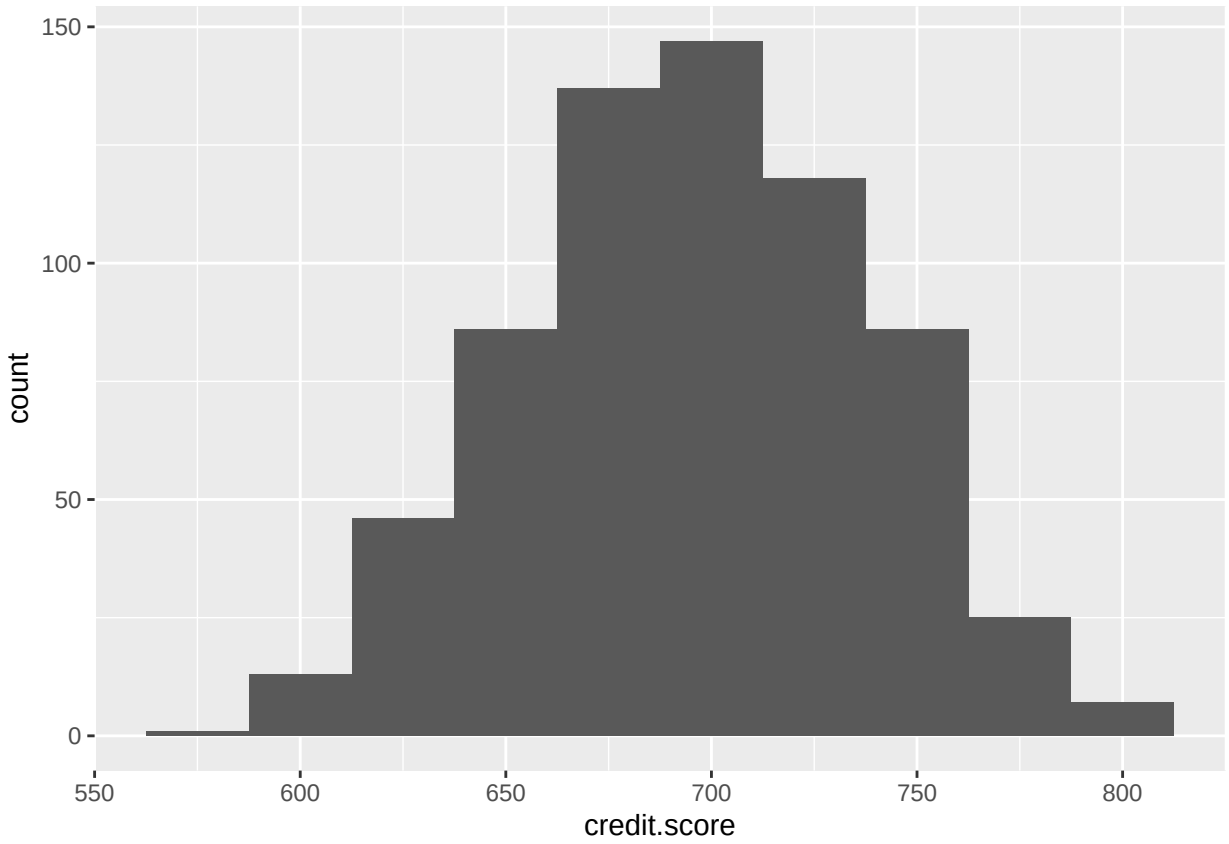
```r
summary(loan)
```

```
##       ...1           cont1           cont2            cont3
##  Min.   :  1.0   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:173.2   1st Qu.:22.60   1st Qu.: 1.000   1st Qu.: 0.165
##  Median :345.5   Median :28.46   Median : 2.750   Median : 1.000
##  Mean   :345.5   Mean   :31.57   Mean   : 4.759   Mean   : 2.223
##  3rd Qu.:517.8   3rd Qu.:38.23   3rd Qu.: 7.207   3rd Qu.: 2.625
##  Max.   :690.0   Max.   :80.25   Max.   :28.000   Max.   :28.500
##                  NA's   :12
##    bool1           bool2            cont4        bool3             cont5
##  Mode :logical   Mode :logical   Min.   : 0.0   Mode :logical   Min.   :   0
##  FALSE:329       FALSE:395       1st Qu.: 0.0   FALSE:374       1st Qu.:  75
##  TRUE :361       TRUE :295       Median : 0.0   TRUE :316       Median : 160
##                                  Mean   : 2.4                   Mean   : 184
##                                  3rd Qu.: 3.0                   3rd Qu.: 276
##                                  Max.   :67.0                   Max.   :2000
##                                                                 NA's   :13
##      cont6            approval          credit.score        ages
##  Min.   :     0.0   Length:690         Min.   :583.7   Min.   :11.00
##  1st Qu.:     0.0   Class :character   1st Qu.:666.7   1st Qu.:31.00
##  Median :     5.0   Mode  :character   Median :697.3   Median :38.00
##  Mean   :  1017.4                      Mean   :696.4   Mean   :39.67
##  3rd Qu.:   395.5                      3rd Qu.:726.4   3rd Qu.:48.00
##  Max.   :100000.0                      Max.   :806.0   Max.   :84.00
##
```

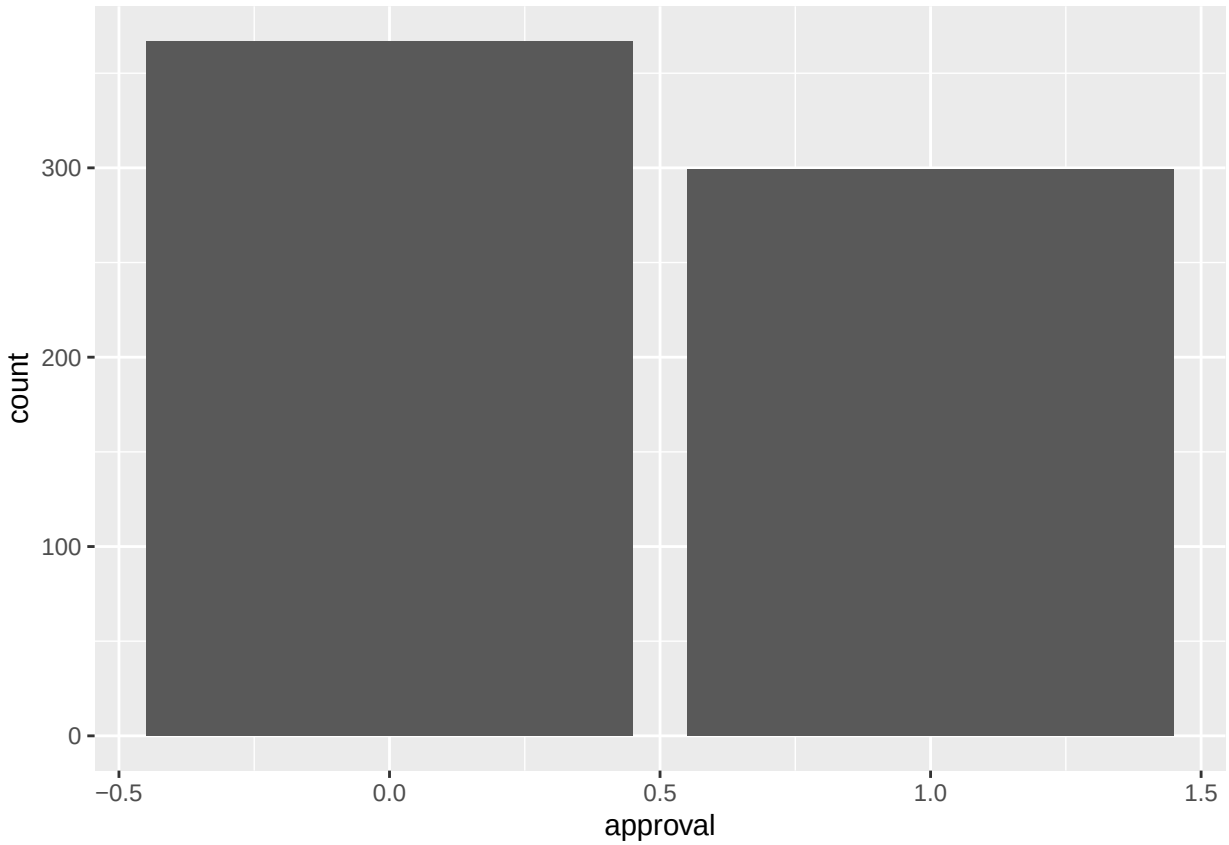
```r
loan <- column_to_rownames(loan, var = "...1")
head(loan)
```

```
##   cont1 cont2 cont3 bool1 bool2 cont4 bool3 cont5 cont6 approval credit.score
## 1 30.83 0.000  1.25  TRUE  TRUE     1 FALSE   202     0        +       664.60
## 2 58.67 4.460  3.04  TRUE  TRUE     6 FALSE    43   560        +       693.88
## 3 24.50 0.500  1.50  TRUE FALSE     0 FALSE   280   824        +       621.82
## 4 27.83 1.540  3.75  TRUE  TRUE     5  TRUE   100     3        +       653.97
## 5 20.17 5.625  1.71  TRUE FALSE     0 FALSE   120     0        +       670.26
## 6 32.08 4.000  2.50  TRUE FALSE     0  TRUE   360     0        +       672.16
##   ages
## 1   42
## 2   54
## 3   29
## 4   58
## 5   65
## 6   61
```

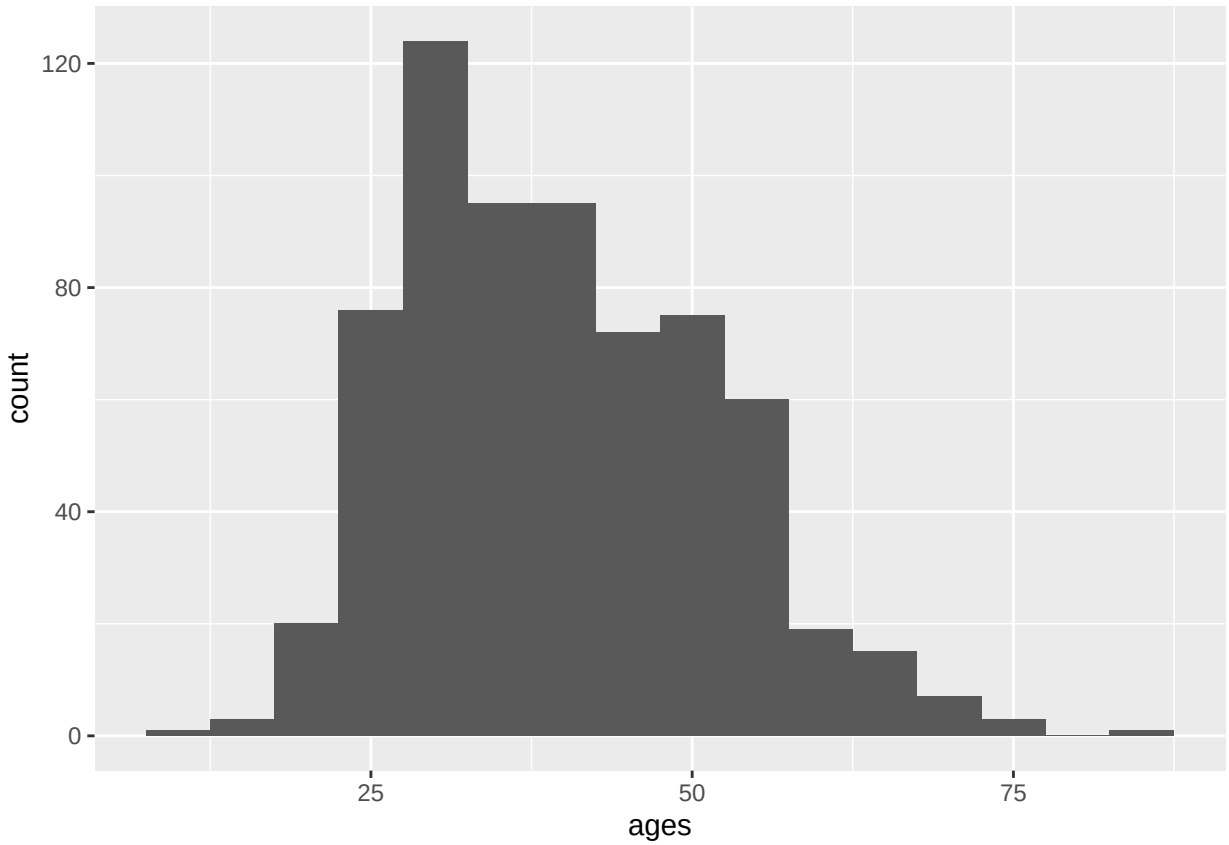
```r
# drop na values
loan <- loan %>% drop_na()

loan$approval <- as.integer(loan$approval == "+")
loan[,c("bool1", "bool2", "bool3")] <- ifelse(loan[,c("bool1", "bool2", "bool3")] == "TRUE", 1, 0)

ggplot(loan, aes(credit.score)) + geom_histogram(binwidth = 25)
```



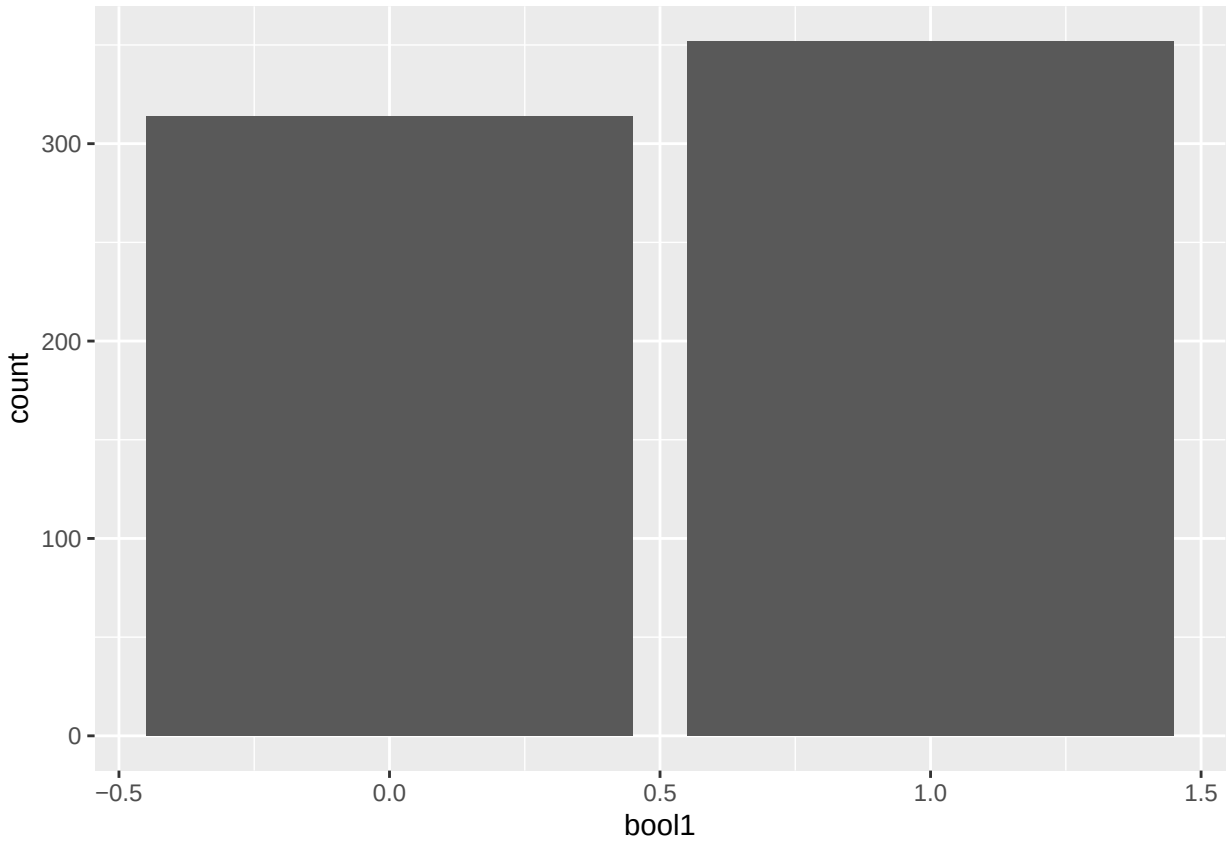

```
ggplot(loan, aes(approval)) + geom_bar()
```



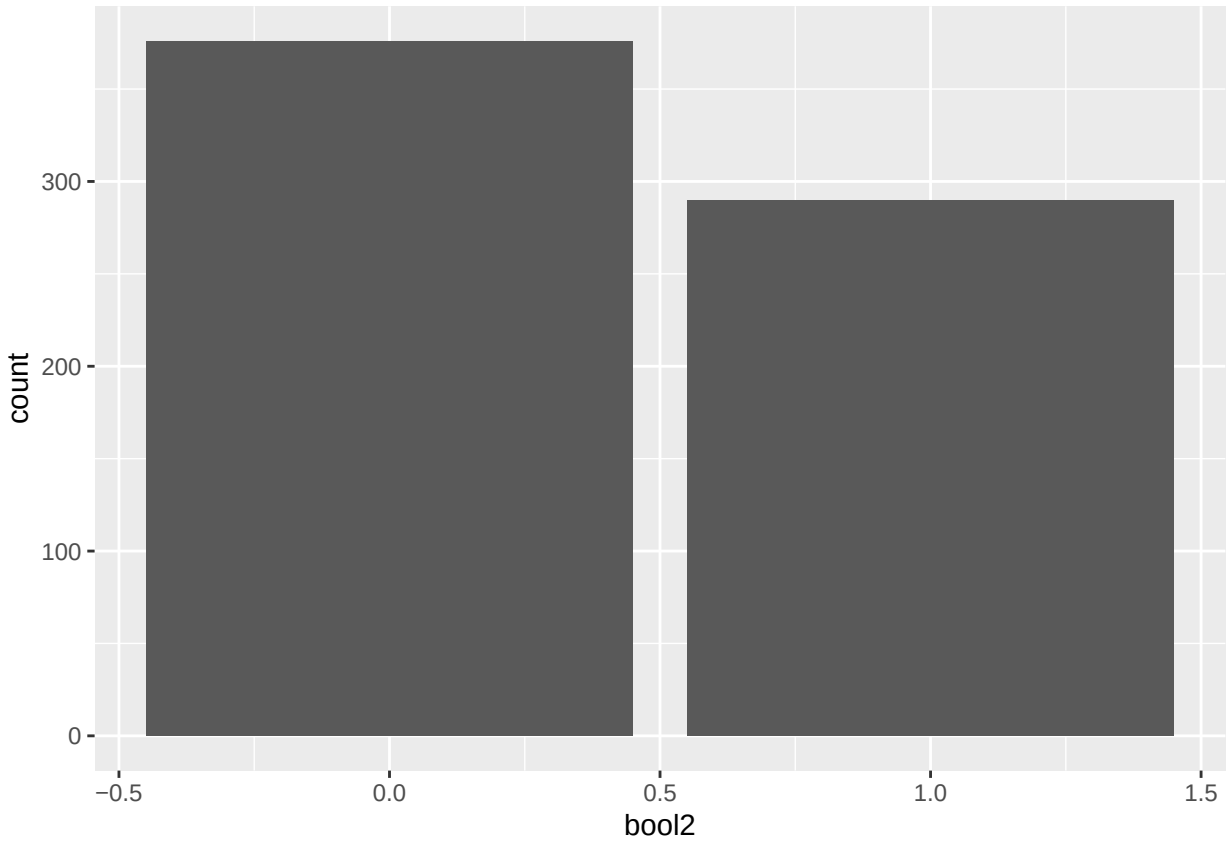

```
ggplot(loan, aes(ages)) + geom_histogram(binwidth = 5)
```



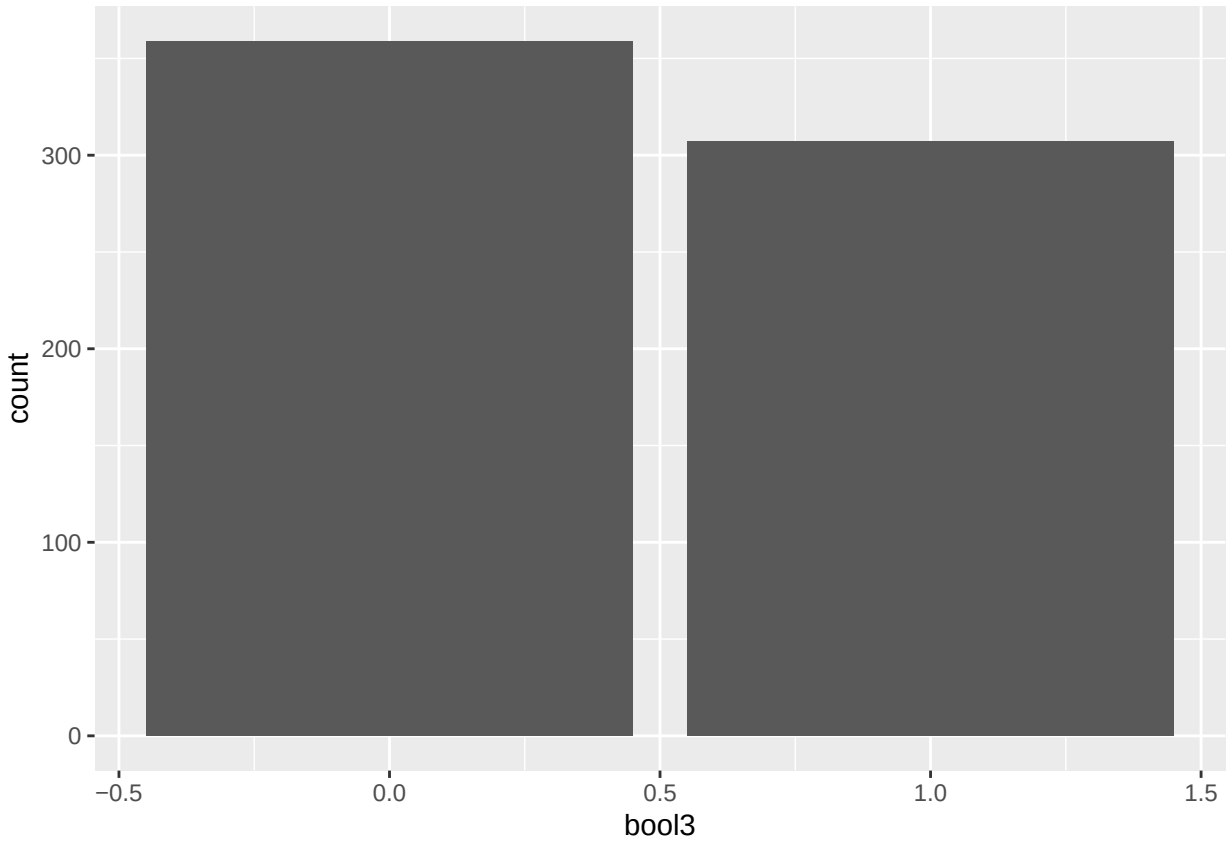

```
ggplot(loan, aes(bool1)) + geom_bar()
```



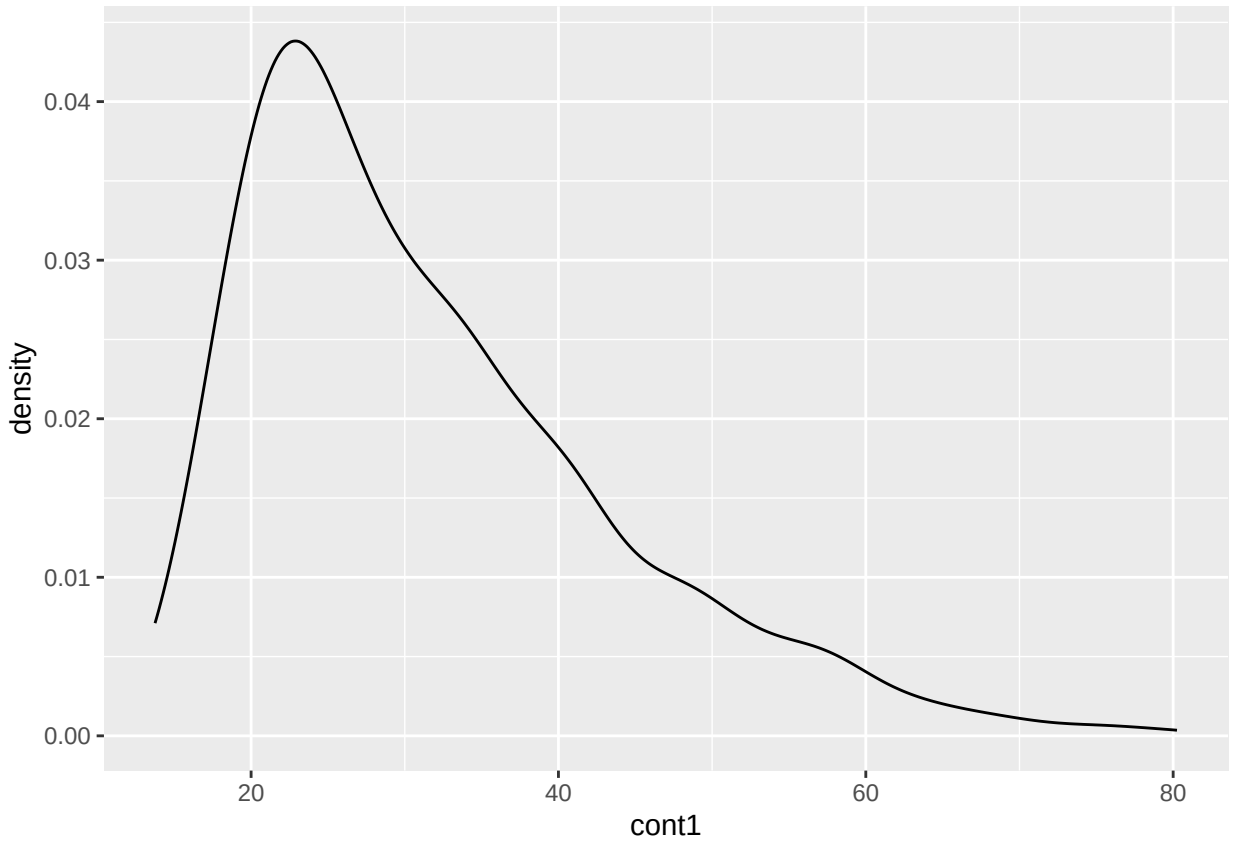

```
ggplot(loan, aes(bool2)) + geom_bar()
```



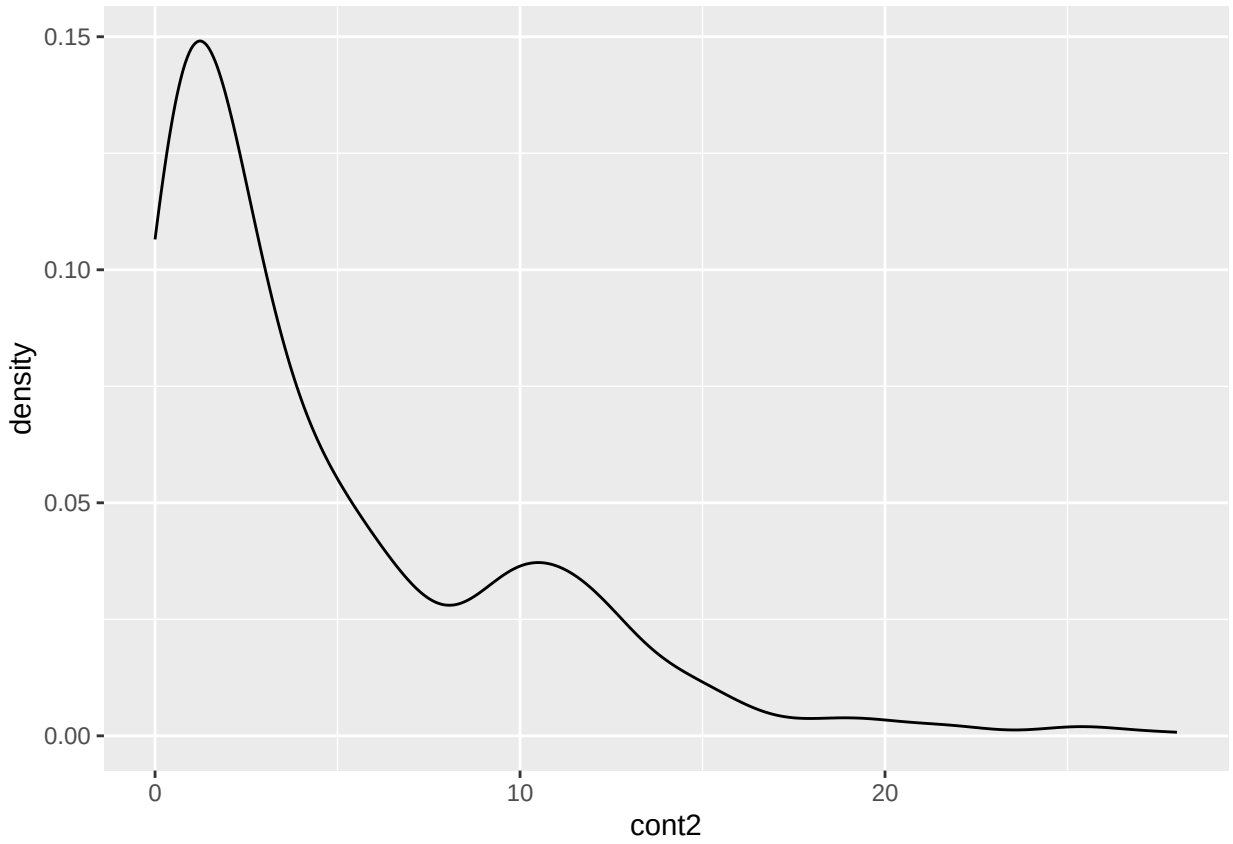

```
ggplot(loan, aes(bool3)) + geom_bar()
```



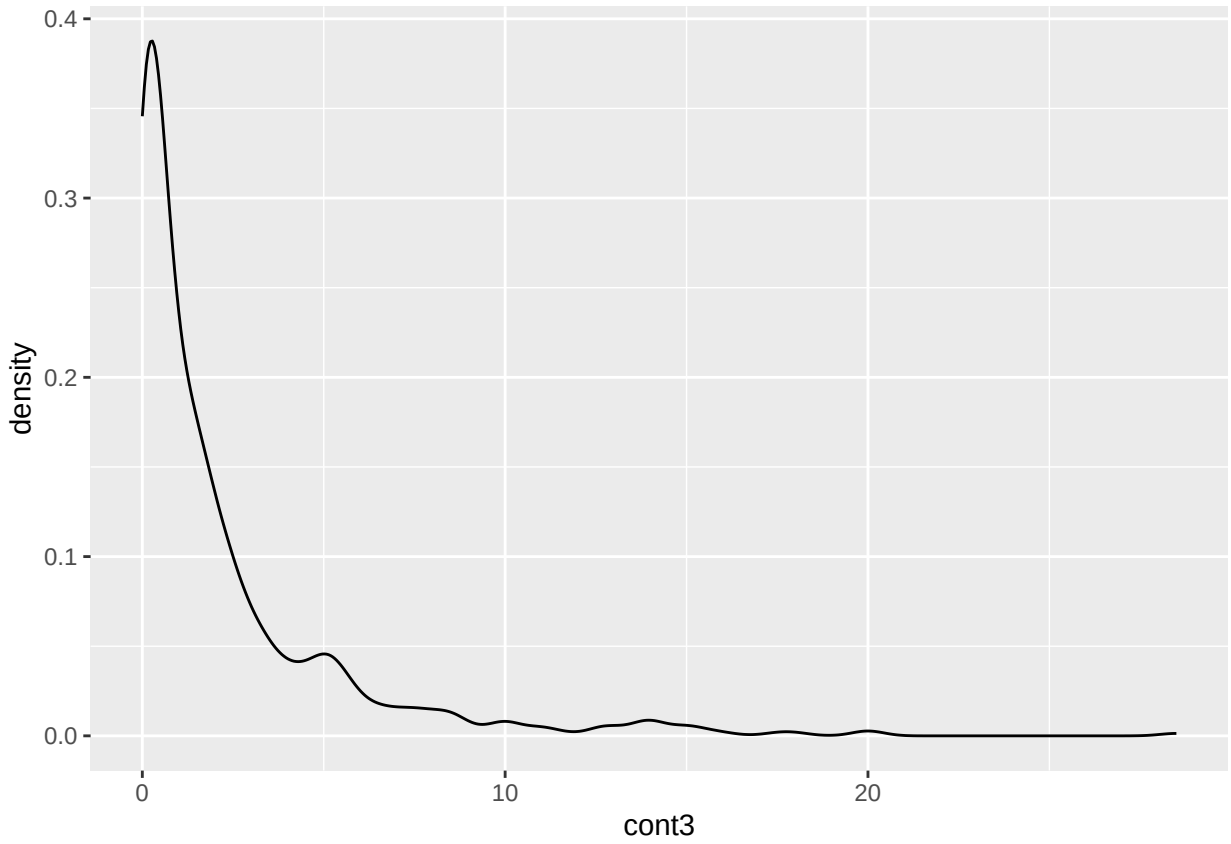

```
ggplot(loan, aes(cont1)) + geom_density()
```



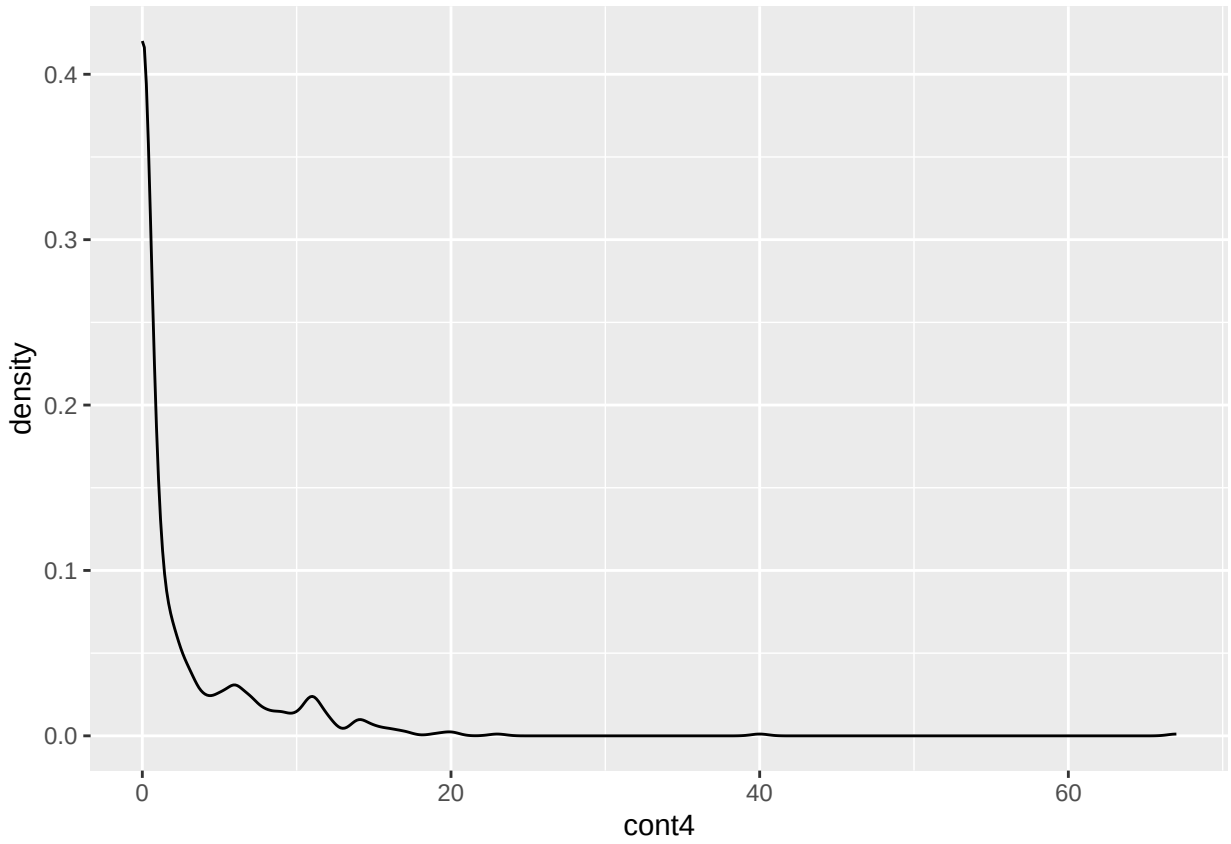

```
ggplot(loan, aes(cont2)) + geom_density()
```



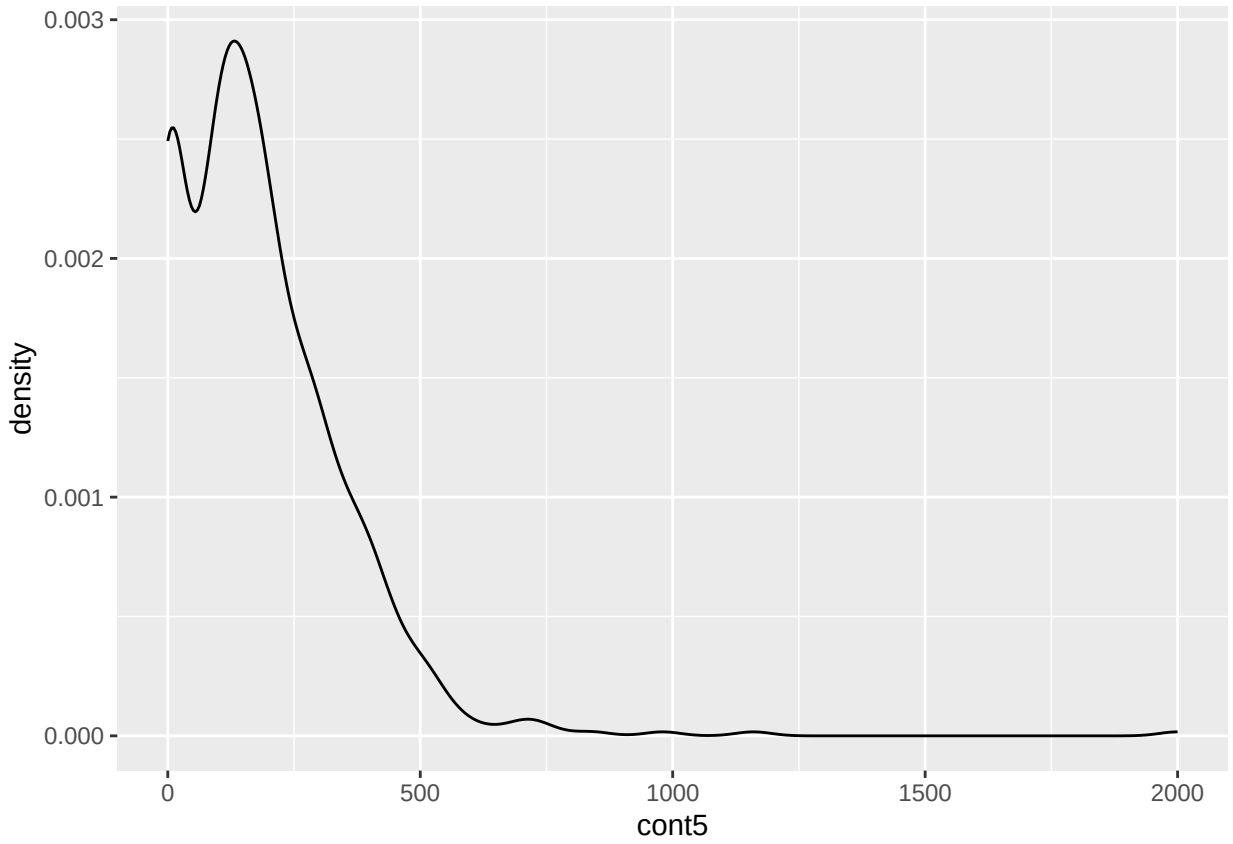

```
ggplot(loan, aes(cont3)) + geom_density()
```



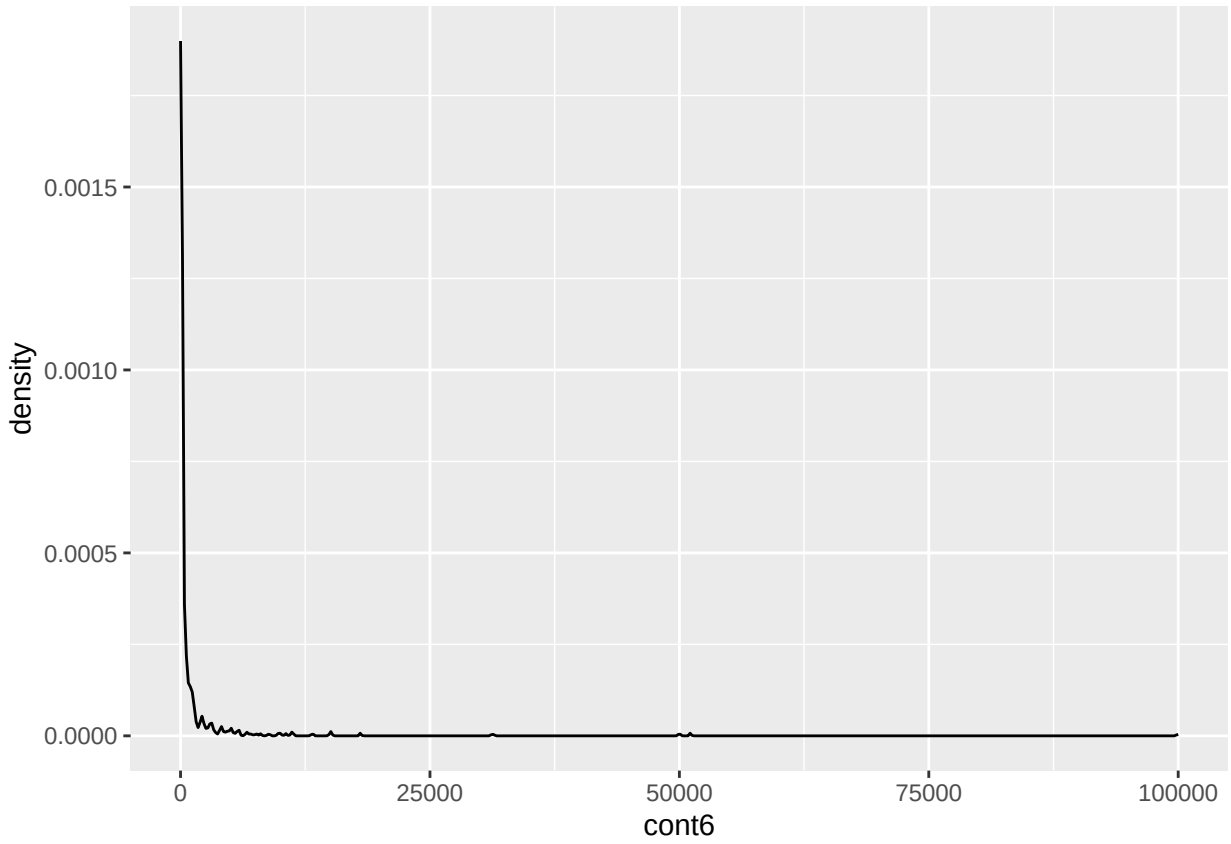

```
ggplot(loan, aes(cont4)) + geom_density()
```




```
ggplot(loan, aes(cont5)) + geom_density()
```




```
ggplot(loan, aes(cont6)) + geom_density()
```



- b: Now apply normalization to some of these numerical distributions. Specifically, choose to apply z-score to one, min-max to another, and decimal scaling to a third. Explain your choices of which normalization applies to which variable in terms of what the variable means, what distribution it starts with, and how the normalization will affect it.

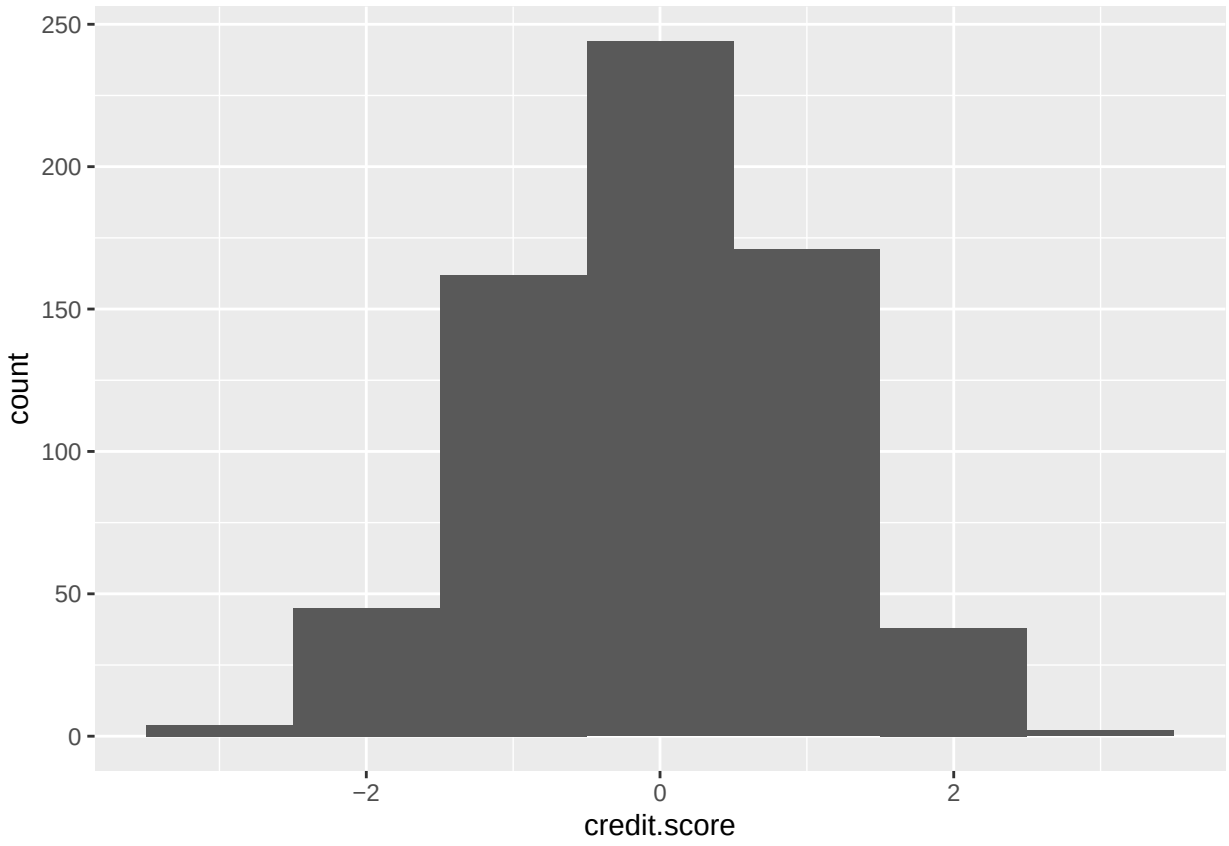
```
# z-score standardization
# formula: Z = (X - mean) / sd, where X is a single raw data value, mean is the population mean, and sd
copy_loan <- loan
mean_credit <- mean(copy_loan$credit.score)
sd_cred <- sd(copy_loan$credit.score)
copy_loan$credit.score <- (copy_loan$credit.score - mean_credit) / sd_cred

# min-max normalization
# formula: (x - min(x)) / (max(x) - min(x)) * (new_max - new_min) + new_max, where x is a single raw da
copy_loan$cont5 <- ((copy_loan$cont5 - min(copy_loan$cont5)) / (max(copy_loan$cont5) - min(copy_loan$co

#decimal scaling
# formula: new_value = x/10i, where i is such that the max of |new_value| is less than 1
copy_loan$cont6 <- copy_loan$cont6 / 100000
```

I decided to perform a z-score standardization, or normalization, on the credit score attribute because the distribution for this variable was normal. The scaling would not have interfered with the mean or the distribution of the credit scores. I knew I wanted to scale both cont5 and cont6 as both these variables had data with large values relative to the other variables. As both variables did not have any data below 0, the decimal scaling and min-max normalization with a minimum value of 0 and maximum value of 1 scaled both variables to the same range (from 0 - 1). Therefore, I applied the min-max normalization to cont5 variable and decimal scaling to cont6 variable.

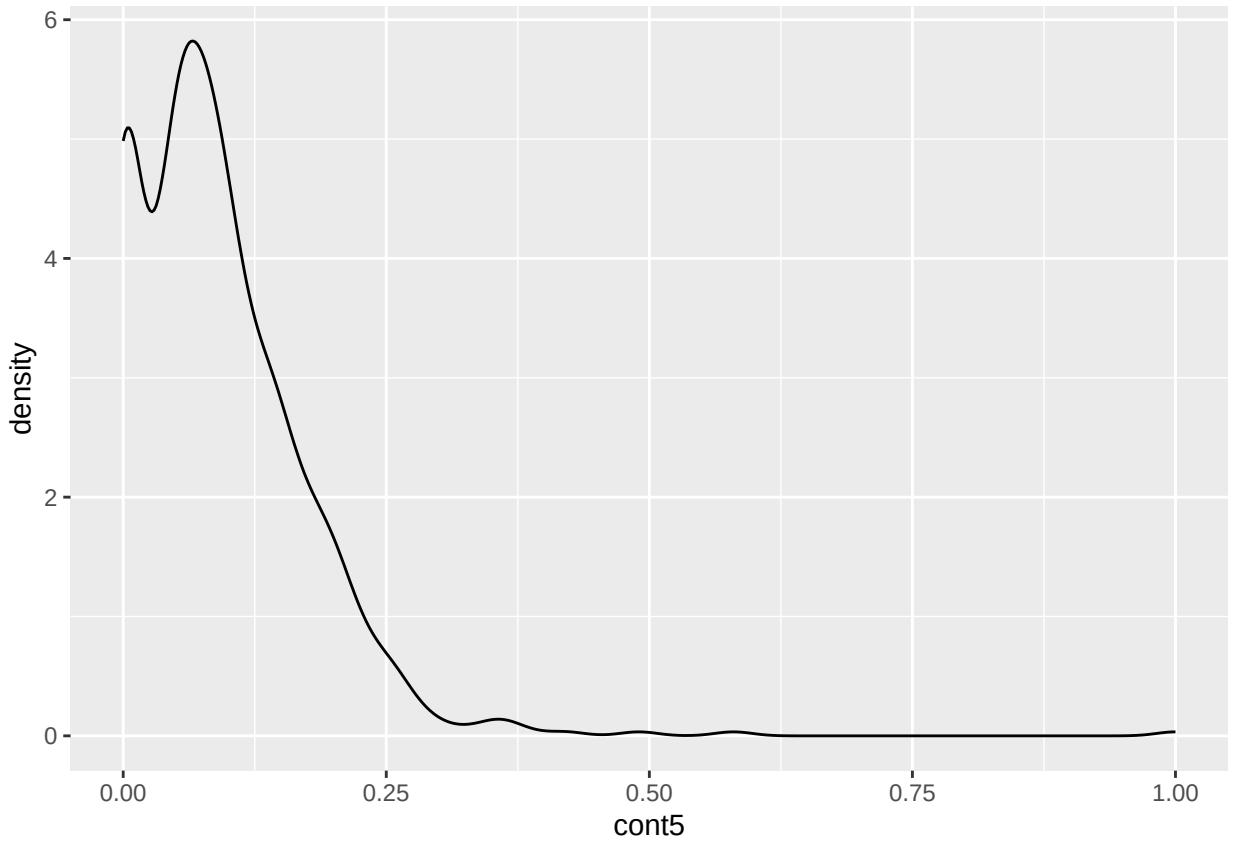
In all instances, the distribution of the attribute that was scaled did not change when compared to the distribution prior to the scaling. Additionally, the normalization scaled the range of values for all the attributes that was substantially different than the value range of the other variables. With the scale, there were less variables that would have more influence on the outcome of a model from having larger values or significant outliers.

- c: Visualize the new distributions for the variables that have been normalized. What has changed from the previous visualization?

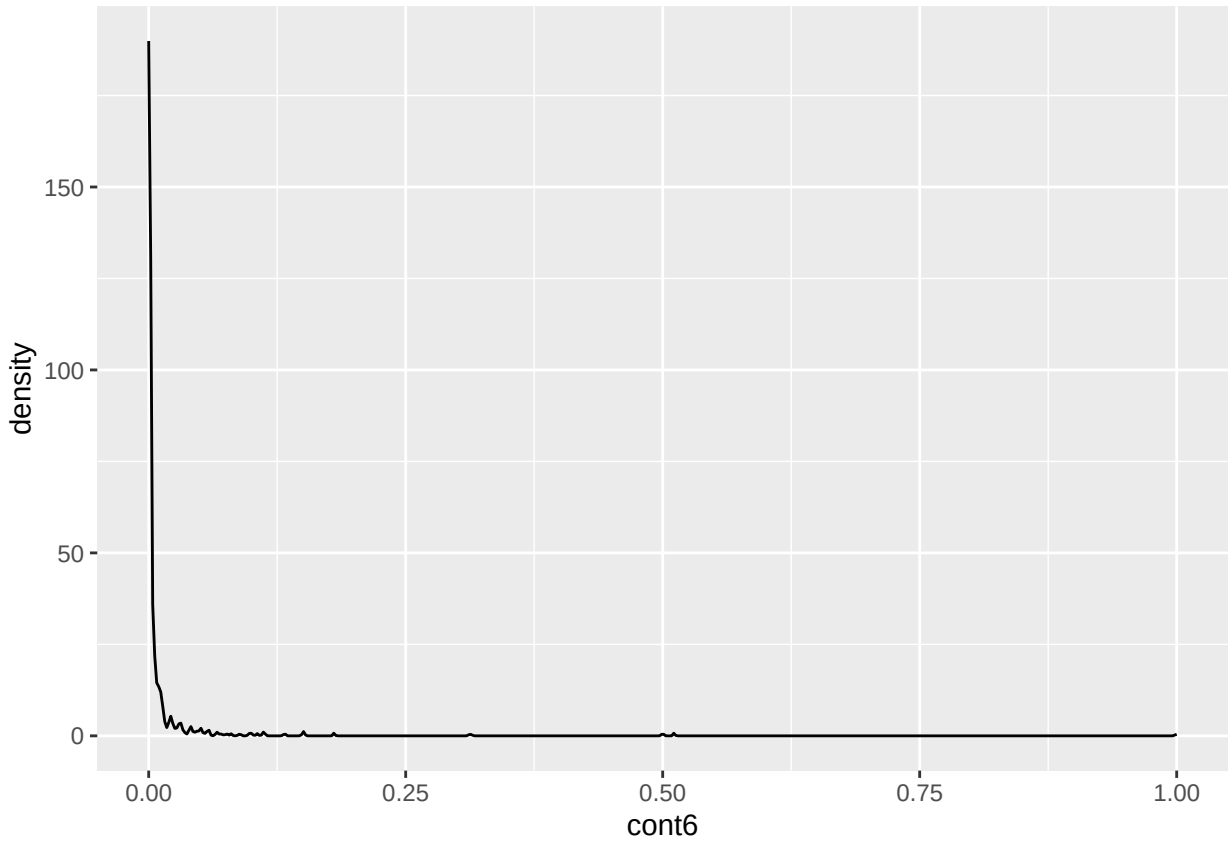
```
# z-score standardization
ggplot(copy_loan, aes(credit.score)) + geom_histogram(binwidth = 1)
```




```
# min-max normalization
ggplot(copy_loan, aes(cont5)) + geom_density()
```




```
# decimal scaling
ggplot(copy_loan, aes(cont6)) + geom_density()
```



In all instances of normalization, the only difference to the visualization from the previous visualization was the data range of each variable, reducing all values from the thousands to the single digits.

- d: Choose one of the numerical variables to work with for this problem. Let's call it v. Create a new variable called v_bins that is a binned version of that variable. This v_bins will have a new set of values like low, medium, high. Choose the actual new values (you don't need to use low, medium, high) and the ranges of v that they represent based on your understanding of v from your visualizations. You can use equal depth, equal width or custom ranges. Explain your choices: why did you choose to create that number of values and those particular ranges?

```
v <- loan
v_bins <- v %>% mutate(credit.cat = cut(credit.score, breaks = c(300.00, 579.00, 669.00, 739.00, 799.00
head(v_bins)
```

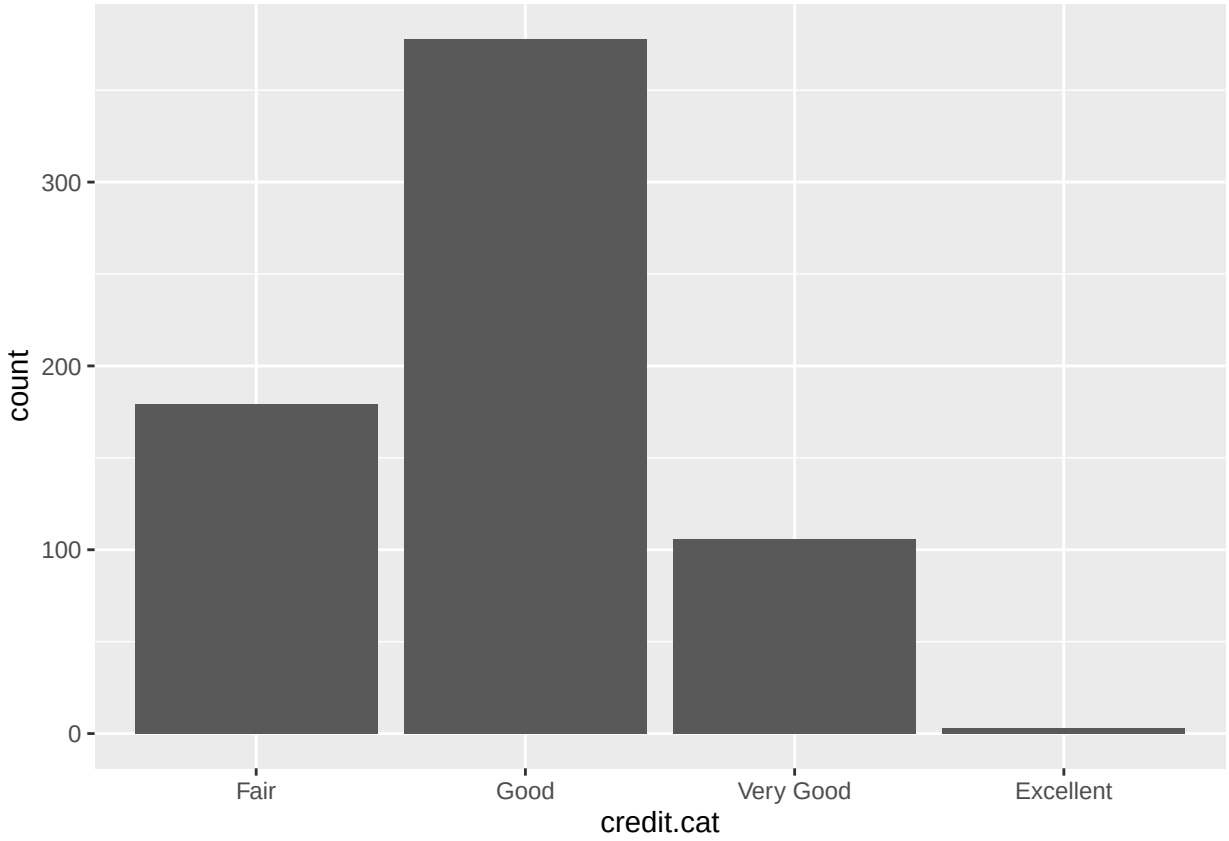
```
##   cont1 cont2 cont3 bool1 bool2 cont4 bool3 cont5 cont6 approval credit.score
## 1 30.83 0.000  1.25     1     1     1     0   202     0        1       664.60
## 2 58.67 4.460  3.04     1     1     6     0    43   560        1       693.88
## 3 24.50 0.500  1.50     1     0     0     0   280   824        1       621.82
## 4 27.83 1.540  3.75     1     1     5     1   100     3        1       653.97
## 5 20.17 5.625  1.71     1     0     0     0   120     0        1       670.26
## 6 32.08 4.000  2.50     1     0     0     1   360     0        1       672.16
##   ages credit.cat
## 1   42       Fair
## 2   54       Good
## 3   29       Fair
## 4   58       Fair
## 5   65       Good
```

```
## 6   61       Good
```

As many financial institution look at loan applicant's credit scores and determine whether the score is categorically good or bad, the credit score attribute was categorized into bins that reflected the score for that category. The credit.cat was created to store the bins for these scores. The particular ranges for each bin was pre-determined by common practices in the financial industry.

- e: Building on (d), use v_bins to create a smoothed version of v. Choose a smoothing strategy to create a numerical version of the binned variable and explain your choices.

```
# find out how many bins there are
ggplot(v_bins, aes(credit.cat)) + geom_bar()
```




```
fair <- v_bins %>% filter(credit.cat == "Fair") %>% mutate(credit.score = mean(credit.score, na.rm = TRU
good <- v_bins %>% filter(credit.cat == "Good") %>% mutate(credit.score = mean(credit.score, na.rm = TRU
v_good <- v_bins %>% filter(credit.cat == "Very Good") %>% mutate(credit.score = mean(credit.score, na.
excel <- v_bins %>% filter(credit.cat == "Excellent") %>% mutate(credit.score = mean(credit.score, na.rm

new_v_bins <- bind_rows(list(fair, good, v_good, excel))
head(new_v_bins)
```

```
##    cont1 cont2 cont3 bool1 bool2 cont4 bool3 cont5 cont6 approval credit.score
## 1  30.83 0.000 1.250     1     1     1     0   202     0        1     644.5341
## 3  24.50 0.500 1.500     1     0     0     0   280   824        1     644.5341
## 4  27.83 1.540 3.750     1     1     5     1   100     3        1     644.5341
```

```
## 13 38.25 6.000 1.000     1     0     0     1     0     0        1     644.5341
## 16 36.67 4.415 0.250     1     1    10     1   320     0        1     644.5341
## 19 21.83 0.250 0.665     1     0     0     1     0     0        1     644.5341
##    ages credit.cat
## 1    42       Fair
## 3    29       Fair
## 4    58       Fair
## 13   60       Fair
## 16   53       Fair
## 19   65       Fair
```

For each of the existing bins, the mean score was determined and used to replace their credit score. This will not only smooth the bins, but also help visualize the average for each category.

Problem 2

This is the first homework problem using machine learning algorithms. You will perform a straightforward training and evaluation of a support vector machine on the bank data from Problem 1. Start with a fresh copy, but be sure to remove rows with missing values first.

```
svm_data <- loan
head(svm_data)
```

```
##   cont1 cont2 cont3 bool1 bool2 cont4 bool3 cont5 cont6 approval credit.score
## 1 30.83 0.000  1.25     1     1     1     0   202     0        1       664.60
## 2 58.67 4.460  3.04     1     1     6     0    43   560        1       693.88
## 3 24.50 0.500  1.50     1     0     0     0   280   824        1       621.82
## 4 27.83 1.540  3.75     1     1     5     1   100     3        1       653.97
## 5 20.17 5.625  1.71     1     0     0     0   120     0        1       670.26
## 6 32.08 4.000  2.50     1     0     0     1   360     0        1       672.16
##   ages
## 1   42
## 2   54
## 3   29
## 4   58
## 5   65
## 6   61
```

- a: Apply SVM to the data from Problem 1 to predict approval and report the accuracy using 10-fold cross validation.

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
svm_data$approval <- as.factor(svm_data$approval)

cv_method <- trainControl(method = "cv", number = 10)
preproc <- c("center", "scale")

library(e1071)
cv_svm <- train(approval ~., data = svm_data, method = "svmLinear", trControl = cv_method, preProcess =
cv_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 666 samples
##  11 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 599, 600, 599, 599, 599, 599, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8633573  0.7283773
##
## Tuning parameter 'C' was held constant at a value of 1
```

The accuracy of SVM was 86.3% using 10-fold cross validation.

- b: Next, use the grid search functionality when training to optimize the C parameter of the SVM. What parameter was chosen and what is the accuracy?

```r
# First we define the set of values to use for each parameter. SVM has only one: C.
grid <- expand.grid(C = 10^seq(-5, 1, 0.5))

# Fit the model with grid search
svm_grid <- train(approval ~., data = svm_data, method = "svmLinear", trControl = cv_method, tuneGrid =
# View grid search result
svm_grid
```

```
## Support Vector Machines with Linear Kernel
##
## 666 samples
##  11 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 600, 599, 600, 599, 599, 599, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.5510403  0.00000000
##   3.162278e-05  0.5510403  0.00000000
```

```
##   1.000000e-04  0.5510403  0.00000000
##   3.162278e-04  0.5615332  0.02554408
##   1.000000e-03  0.8363863  0.66552798
##   3.162278e-03  0.8649480  0.73181875
##   1.000000e-02  0.8634555  0.72877509
##   3.162278e-02  0.8634555  0.72877509
##   1.000000e-01  0.8634555  0.72877509
##   3.162278e-01  0.8634555  0.72877509
##   1.000000e+00  0.8634555  0.72877509
##   3.162278e+00  0.8634555  0.72877509
##   1.000000e+01  0.8634555  0.72877509
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.003162278.
```

The value for parameter C that was chosen was 0.00316 and this parameter value had an accuracy of 0.86485 with a kappa value of 0.73140.

- c: Sometimes even if the grid of parameters in (b) includes the default value of C = 1 (used in (a)), the accuracy result will be different for this value of C. What could make that different?

The difference between the SVM accuracy from the default parameter value of 1 in the grid search vs. without the grid search can be explained by the difference in sample contained in each of the folds. As the folds in the grid search vs. without grid search may contain different set of samples, the model trained using different sets samples within each folds for the two scenarios (grid vs. non-grid).

Problem 3

We will take SVM further in this problem, showing how it often gets used even when the data are not suitable, by first engineering the numerical features we need. There is a Star Wars dataset in the *dplyr* library. Load that library and you will be able to see it *(head(starwars))*. There are some variables we will not use, so first remove *films, vehicles, starships* and *name.* Also remove rows with missing values

```
data(starwars)
star_data <- select(starwars, -c("films", "vehicles", "starships", "name"))
head(star_data)
```

```
## # A tibble: 6 x 10
##   height  mass hair_color skin_color eye_color birth_year sex   gender homeworld
##    <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>
## 1    172    77 blond      fair       blue            19   male  mascu~ Tatooine
## 2    167    75 <NA>       gold       yellow         112   none  mascu~ Tatooine
## 3     96    32 <NA>       white, bl~ red             33   none  mascu~ Naboo
## 4    202   136 none       white      yellow          41.9 male  mascu~ Tatooine
## 5    150    49 brown      light      brown           19   fema~ femin~ Alderaan
## 6    178   120 brown, gr~ light      blue            52   male  mascu~ Tatooine
## # i 1 more variable: species <chr>
```

```
summary(star_data)
```

```
##      height           mass          hair_color         skin_color
##  Min.   : 66.0   Min.   :  15.00   Length:87          Length:87
##  1st Qu.:167.0   1st Qu.:  55.60   Class :character   Class :character
##  Median :180.0   Median :  79.00   Mode  :character   Mode  :character
##  Mean   :174.6   Mean   :  97.31
##  3rd Qu.:191.0   3rd Qu.:  84.50
##  Max.   :264.0   Max.   :1358.00
##  NA's   :6       NA's   :28
##   eye_color           birth_year         sex              gender
##  Length:87          Min.   :  8.00   Length:87          Length:87
##  Class :character   1st Qu.: 35.00   Class :character   Class :character
##  Mode  :character   Median : 52.00   Mode  :character   Mode  :character
##                     Mean   : 87.57
##                     3rd Qu.: 72.00
##                     Max.   :896.00
##                     NA's   :44
##   homeworld           species
##  Length:87          Length:87
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##
```

```
# remove rows with missing values
star_data <- drop_na(star_data)
dim(star_data)
```

```
## [1] 29 10
```

```
summary(star_data)
```

```
##      height           mass          hair_color         skin_color
##  Min.   : 88.0   Min.   : 20.00   Length:29          Length:29
##  1st Qu.:172.0   1st Qu.: 75.00   Class :character   Class :character
##  Median :180.0   Median : 79.00   Mode  :character   Mode  :character
##  Mean   :178.7   Mean   : 77.77
##  3rd Qu.:188.0   3rd Qu.: 83.00
##  Max.   :228.0   Max.   :136.00
##   eye_color           birth_year         sex              gender
##  Length:29          Min.   :  8.00   Length:29          Length:29
##  Class :character   1st Qu.: 31.00   Class :character   Class :character
##  Mode  :character   Median : 46.00   Mode  :character   Mode  :character
##                     Mean   : 51.29
##                     3rd Qu.: 57.00
##                     Max.   :200.00
##   homeworld           species
##  Length:29          Length:29
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

- a: Several variables are categorical. We will use dummy variables to make it possible for SVM to use these. Leave the *gender* category out of the dummy variable conversion to use as a categorical for prediction. Show the resulting *head*.

```
library(fastDummies)
star_data$sex_male <- ifelse(star_data$sex == "male", 1, 0)
star_data$sex_female <- ifelse(star_data$sex == "female", 1, 0)

dummy_star <- dummy_cols(star_data, select_columns = c("eye_color", "hair_color", "skin_color", "homewo
head(dummy_star)
```

```
## # A tibble: 6 x 73
##   height  mass hair_color skin_color eye_color birth_year sex   gender homeworld
##    <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>
## 1    172    77 blond      fair       blue            19   male  mascu~ Tatooine
## 2    202   136 none       white      yellow          41.9 male  mascu~ Tatooine
## 3    150    49 brown      light      brown           19   fema~ femin~ Alderaan
## 4    178   120 brown, gr~ light      blue            52   male  mascu~ Tatooine
## 5    165    75 brown      light      blue            47   fema~ femin~ Tatooine
## 6    183    84 black      light      brown           24   male  mascu~ Tatooine
## # i 64 more variables: species <chr>, sex_male <dbl>, sex_female <dbl>,
## #   eye_color_black <int>, eye_color_blue <int>, `eye_color_blue-gray` <int>,
## #   eye_color_brown <int>, eye_color_hazel <int>, eye_color_orange <int>,
## #   eye_color_red <int>, eye_color_yellow <int>,
## #   `hair_color_auburn, white` <int>, hair_color_black <int>,
## #   hair_color_blond <int>, hair_color_brown <int>,
## #   `hair_color_brown, grey` <int>, hair_color_grey <int>, ...
```

- b: Use SVM to predict gender and report the accuracy.

```
# remove non-numerical columns
new_star_data <- select(dummy_star, -c("eye_color", "hair_color", "skin_color", "homeworld", "species",
head(new_star_data)
```

```
## # A tibble: 6 x 67
##   height  mass birth_year gender    sex_male sex_female eye_color_black
##    <int> <dbl>      <dbl> <chr>        <dbl>      <dbl>           <int>
## 1    172    77       19   masculine        1          0               0
## 2    202   136       41.9 masculine        1          0               0
## 3    150    49       19   feminine         0          1               0
## 4    178   120       52   masculine        1          0               0
## 5    165    75       47   feminine         0          1               0
## 6    183    84       24   masculine        1          0               0
## # i 60 more variables: eye_color_blue <int>, `eye_color_blue-gray` <int>,
## #   eye_color_brown <int>, eye_color_hazel <int>, eye_color_orange <int>,
## #   eye_color_red <int>, eye_color_yellow <int>,
## #   `hair_color_auburn, white` <int>, hair_color_black <int>,
## #   hair_color_blond <int>, hair_color_brown <int>,
## #   `hair_color_brown, grey` <int>, hair_color_grey <int>,
## #   hair_color_none <int>, hair_color_white <int>, skin_color_blue <int>, ...
```

```r
library(e1071)
train_control <- trainControl(method = "cv", number = 10)

star_svm <- train(gender ~., data = new_star_data, method = "svmLinear", trControl = train_control)
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```r
star_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 29 samples
## 66 predictors
##  2 classes: 'feminine', 'masculine'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 26, 27, 26, 27, 26, 26, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.95      0.7777778
##
## Tuning parameter 'C' was held constant at a value of 1
```

The accuracy of the SVM was 91.66% with a kappa value of 0.7142.

- c: Given that we have so many variables, it makes sense to consider using PCA. Run PCA on the data and determine an appropriate number of components to use. Document how you made the decision, including any graphs you used. Create a reduced version of the data with that number of principle components. **Note: make sure to remove gender from the data before running PCA because it would be cheating if PCA had access to the label you will use.** *Add it back in after reducing the data and show the result.*

```r
# Under-represented columns might cause issues for PCA, called near zero variance problem. Any underrep
nzv <- nearZeroVar(new_star_data)
nzv
```

```
##  [1]  7  9 13 15 19 20 23 24 25 28 32 33 34 35 37 38 39 40 42 43 44 45 46 47 49
## [26] 51 52 53 54 56 57 58 59 61 63 64 65 66 67
```

```r
# new data for pca without near zero variance columns and target column
pca_star_data <- select(new_star_data, -c(nzv, "gender"))
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use ‘all_of()‘ or ‘any_of()‘ instead.
##   # Was:
##   data %>% select(nzv)
##
##   # Now:
##   data %>% select(all_of(nzv))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```r
head(pca_star_data)
```

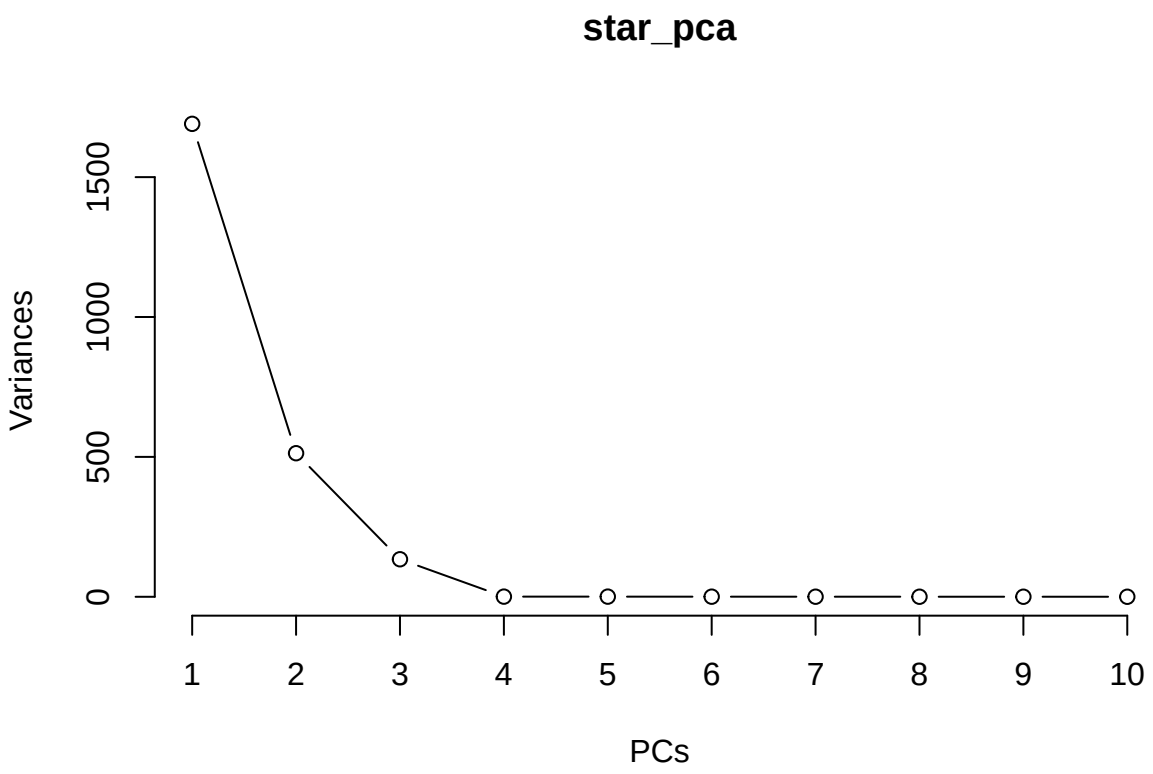
```
## # A tibble: 6 x 27
##   height  mass birth_year sex_male sex_female eye_color_blue eye_color_brown
##    <int> <dbl>      <dbl>    <dbl>      <dbl>          <int>           <int>
## 1    172    77       19           1          0              1               0
## 2    202   136       41.9         1          0              0               0
## 3    150    49       19           0          1              0               1
## 4    178   120       52           1          0              1               0
## 5    165    75       47           0          1              1               0
## 6    183    84       24           1          0              0               1
## # i 20 more variables: eye_color_hazel <int>, eye_color_orange <int>,
## #   eye_color_yellow <int>, hair_color_black <int>, hair_color_blond <int>,
## #   hair_color_brown <int>, hair_color_none <int>, hair_color_white <int>,
## #   skin_color_dark <int>, skin_color_fair <int>, skin_color_light <int>,
## #   skin_color_orange <int>, skin_color_pale <int>, skin_color_yellow <int>,
## #   homeworld_Corellia <int>, homeworld_Mirial <int>, homeworld_Naboo <int>,
## #   homeworld_Tatooine <int>, species_Human <int>, species_Mirialan <int>
```

```r
# PCA
star_pca <- prcomp(pca_star_data)
summary(star_pca)
```

```
## Importance of components:
##                             PC1     PC2      PC3     PC4     PC5     PC6     PC7
## Standard deviation     41.1141 22.6529 11.58095 0.74460 0.72454 0.57667 0.55619
## Proportion of Variance  0.7223  0.2193  0.05731 0.00024 0.00022 0.00014 0.00013
## Cumulative Proportion   0.7223  0.9415  0.99885 0.99909 0.99931 0.99945 0.99959
##                             PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.46320 0.42851 0.33911 0.29923 0.26805 0.24993 0.24346
## Proportion of Variance 0.00009 0.00008 0.00005 0.00004 0.00003 0.00003 0.00003
## Cumulative Proportion  0.99968 0.99976 0.99981 0.99984 0.99987 0.99990 0.99993
##                            PC15    PC16    PC17    PC18    PC19   PC20    PC21
## Standard deviation     0.20585 0.18687 0.17475 0.14552 0.11647 0.1016 0.08663
## Proportion of Variance 0.00002 0.00001 0.00001 0.00001 0.00001 0.0000 0.00000
## Cumulative Proportion  0.99994 0.99996 0.99997 0.99998 0.99999 1.0000 1.00000
```

```
##                            PC22    PC23    PC24      PC25      PC26      PC27
## Standard deviation     0.07295 0.06009 0.04434 2.685e-15 2.685e-15 2.685e-15
## Proportion of Variance 0.00000 0.00000 0.00000 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.00000 1.00000 1.00000 1.000e+00 1.000e+00 1.000e+00
```

```
# Visualize the scree plot
screeplot(star_pca, type = "l") + title(xlab = "PCs")
```




```
## integer(0)
```

```
# Create the components
preProc <- preProcess(pca_star_data, method="pca", pcaComp=3)
star_pc <- predict(preProc, pca_star_data)

# Put back target column
star_pc$gender <- as.factor(new_star_data$gender)

# New PCs are predictors for gender
(star_pc)
```

```
##           PC1         PC2          PC3    gender
## 1   0.6858547 -1.80484940 -3.323464575 masculine
## 2   2.2495969  1.29566292 -1.807045684 masculine
## 3  -1.8759412 -2.70627054  2.288013364  feminine
## 4   0.6800513 -0.80337159 -2.380616847 masculine
## 5  -1.7574602 -2.09005731 -0.534506569  feminine
## 6   0.1756673 -1.70476033 -0.737179801 masculine
## 7   0.9695630 -0.55444932 -0.526388937 masculine
```

```
## 8   0.9237893 -1.48113954 -3.672382387 masculine
## 9   0.9433901  1.56766949 -1.719815162 masculine
## 10  1.0224563 -2.83747546  0.881799422 masculine
## 11  0.8308188 -2.69557750  1.227709587 masculine
## 12  1.5177353  1.80214954  0.534457779 masculine
## 13  0.4097426 -1.33616618 -0.254972391 masculine
## 14  1.1941188  1.63909296 -0.075848230 masculine
## 15  0.3106727 -0.93139012  0.381393828 masculine
## 16  0.3335691 -0.12496316 -0.479629295 masculine
## 17  1.0292808  2.19320274  1.239187630 masculine
## 18 -1.0005480 -1.93234995  3.035795434 masculine
## 19 -1.5093518 -1.86795821  2.653740094  feminine
## 20  1.3368363  3.55018589  2.790182314 masculine
## 21  1.1229362  1.99913277  0.395040893 masculine
## 22 -1.2994785  0.65553343  2.042596596  feminine
## 23  1.3421577  0.08873486  0.598903891 masculine
## 24  1.8436612  2.95455590 -0.575377869 masculine
## 25  1.0149669  2.12929481  1.289465974 masculine
## 26 -7.0844331  2.04544520 -1.408194287  feminine
## 27 -7.2165117  1.86843871 -1.207913718  feminine
## 28  1.5566505 -0.38264011 -0.656824431 masculine
## 29  0.2502088 -0.53568052  0.001873378 masculine
```

```
summary(star_pc)
```

```
##       PC1                PC2                PC3                 gender
##  Min.   :-7.2165   Min.   :-2.8375   Min.   :-3.67238   feminine : 6
##  1st Qu.: 0.1757   1st Qu.:-1.7048   1st Qu.:-0.73718   masculine:23
##  Median : 0.8308   Median :-0.3826   Median :-0.07585
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 1.1229   3rd Qu.: 1.8021   3rd Qu.: 1.22771
##  Max.   : 2.2496   Max.   : 3.5502   Max.   : 3.03579
```

The appropriate number of components to use was 3 components as 99.88% of the cumulative proportion of variance was explained by the first 3 components. The addition of each component after the first 3 components was marginal and did not provide significant value to the model. Additionally, as seen in the scree plot, the elbow of the curve was found at the 3rd component after which the curve significantly flattened.

- d: Use SVM to predict *gender* again, but this time use the data resulting from PCA. Evaluate the results with a confusion matrix and at least two partitioning methods, using grid search on the C parameter each time.

```
grid_pca <- expand.grid(C = 10^seq(-5,2,0.5))

# k-fold cv model fit, train_control = trainControl(method = 'cv', number = 10)
pca_svm <- train(gender ~., data = star_pc, method = "svmLinear", trControl = train_control, tuneGrid =
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
pca_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 29 samples
##  3 predictor
##  2 classes: 'feminine', 'masculine'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 25, 26, 26, 26, 26, 27, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.8083333  0.0000000
##   3.162278e-05  0.8083333  0.0000000
##   1.000000e-04  0.8083333  0.0000000
##   3.162278e-04  0.8083333  0.0000000
##   1.000000e-03  0.8083333  0.0000000
##   3.162278e-03  0.8083333  0.0000000
##   1.000000e-02  0.8083333  0.0000000
##   3.162278e-02  0.8083333  0.0000000
##   1.000000e-01  0.8750000  0.3333333
##   3.162278e-01  0.8416667  0.2857143
##   1.000000e+00  0.9666667  0.8571429
##   3.162278e+00  0.9333333  0.7142857
##   1.000000e+01  0.9333333  0.7142857
##   3.162278e+01  0.9666667  0.8571429
##   1.000000e+02  0.9666667  0.8571429
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.
```

```
pca_svm_predict <- predict(pca_svm, star_pc)
pca_svm_predict
```

```
##  [1] masculine masculine feminine  masculine feminine  masculine masculine
##  [8] masculine masculine masculine masculine masculine masculine masculine
## [15] masculine masculine masculine feminine  feminine  masculine masculine
## [22] feminine  masculine masculine masculine feminine  feminine  masculine
## [29] masculine
## Levels: feminine masculine
```

```
confusionMatrix(star_pc$gender, pca_svm_predict)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  feminine masculine
##   feminine         6         0
##   masculine        1        22
##
```

```
##                Accuracy : 0.9655
##                  95% CI : (0.8224, 0.9991)
##     No Information Rate : 0.7586
##     P-Value [Acc > NIR] : 0.003392
##
##                   Kappa : 0.901
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8571
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9565
##              Prevalence : 0.2414
##          Detection Rate : 0.2069
##    Detection Prevalence : 0.2069
##       Balanced Accuracy : 0.9286
##
##        'Positive' Class : feminine
##
```

```
# bootstrapping
train_control_boot = trainControl(method = "boot", number = 50)
# Fit the model
svm_boot <- train(gender ~., data = star_pc, method = "svmLinear", trControl = train_control_boot, tune
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
svm_boot
```

```
## Support Vector Machines with Linear Kernel
##
## 29 samples
##  3 predictor
##  2 classes: 'feminine', 'masculine'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 29, 29, 29, 29, 29, 29, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.8221151  0.0000000
##   3.162278e-05  0.8221151  0.0000000
##   1.000000e-04  0.8221151  0.0000000
##   3.162278e-04  0.8221151  0.0000000
##   1.000000e-03  0.8221151  0.0000000
##   3.162278e-03  0.8221151  0.0000000
##   1.000000e-02  0.8221151  0.0000000
##   3.162278e-02  0.8702670  0.3703482
##   1.000000e-01  0.8793045  0.4261212
##   3.162278e-01  0.8987160  0.5603929
```

```
##   1.000000e+00  0.9329211  0.7062667
##   3.162278e+00  0.9390245  0.7090073
##   1.000000e+01  0.9437150  0.7541891
##   3.162278e+01  0.9479372  0.7681138
##   1.000000e+02  0.9479372  0.7681138
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 31.62278.
```

```
svm_boot_predict <- predict(svm_boot, star_pc)
svm_boot_predict
```

```
##  [1] masculine masculine feminine  masculine feminine  masculine masculine
##  [8] masculine masculine masculine masculine masculine masculine masculine
## [15] masculine masculine masculine masculine feminine  masculine masculine
## [22] feminine  masculine masculine masculine feminine  feminine  masculine
## [29] masculine
## Levels: feminine masculine
```

```
confusionMatrix(star_pc$gender, svm_boot_predict)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  feminine masculine
##   feminine         6         0
##   masculine        0        23
##
##                Accuracy : 1
##                  95% CI : (0.8806, 1)
##     No Information Rate : 0.7931
##     P-Value [Acc > NIR] : 0.001204
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.2069
##          Detection Rate : 0.2069
##    Detection Prevalence : 0.2069
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : feminine
##
```

```
# train/test splitting
# set the randomizer seed
set.seed(123)
```

```r
# partition data
index = createDataPartition(y=star_pc$gender, p=0.7, list=FALSE)
# training set
train_set = star_pc[index,]
# testing set
test_set = star_pc[-index,]

# fit the model using the training set
svm_train <- train(gender ~., data = train_set, method = "svmLinear", tuneGrid = grid_pca)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```r
svm_train
```

```
## Support Vector Machines with Linear Kernel
##
## 22 samples
##  3 predictor
##  2 classes: 'feminine', 'masculine'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 22, 22, 22, 22, 22, 22, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.7752973  0.0000000
##   3.162278e-05  0.7752973  0.0000000
##   1.000000e-04  0.7752973  0.0000000
##   3.162278e-04  0.7752973  0.0000000
##   1.000000e-03  0.7752973  0.0000000
##   3.162278e-03  0.7752973  0.0000000
##   1.000000e-02  0.7752973  0.0000000
##   3.162278e-02  0.8369639  0.2346977
##   1.000000e-01  0.8922987  0.4861570
##   3.162278e-01  0.9402670  0.7297691
##   1.000000e+00  0.9697763  0.8884824
##   3.162278e+00  0.9898413  0.9446640
##   1.000000e+01  0.9898413  0.9446640
##   3.162278e+01  0.9898413  0.9446640
##   1.000000e+02  0.9898413  0.9446640
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 3.162278.
```

```r
# predict with test set
pred_test <- predict(svm_train, test_set)

confusionMatrix(as.factor(test_set$gender), pred_test)
```

```
## Confusion Matrix and Statistics
```

```
##
##            Reference
## Prediction  feminine masculine
##   feminine         1         0
##   masculine        1         5
##
##                Accuracy : 0.8571
##                  95% CI : (0.4213, 0.9964)
##     No Information Rate : 0.7143
##     P-Value [Acc > NIR] : 0.3605
##
##                   Kappa : 0.5882
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.5000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.8333
##              Prevalence : 0.2857
##          Detection Rate : 0.1429
##    Detection Prevalence : 0.1429
##       Balanced Accuracy : 0.7500
##
##        'Positive' Class : feminine
##
```

```
# manual quick calculation of accuracy: what proportion of predictions match labels
sum(pred_test == test_set$gender) / nrow(test_set)
```

```
## [1] 0.8571429
```

I utilized 3 partitioning methods: 10-fold cross validation, bootstrapping, and train/tests splitting. In all instances of grid search, accuracy was used to select the optimal model parameter C using the largest accuracy value.

1. 10-fold CV: The final value used for the model was C = 1 with accuracy: 0.9655
2. bootstrapping: The final value used for the model was C = 31.62278 with accuracy: 1
3. test/train splitting: The final value used for the model was C = 10 with accuracy: 0.9298. The confusion matrix produced an accuracy of 0.8571.

- e: Whether or not it has improved the accuracy, what has PCA done for the complexity of the model?

PCA is an unsupervised machine learning technique that reduced the number of dimensions in the star wars data set, transforming the original variables that are potentially correlated to each other into a smaller set of components (principal components). Although the interpretability of the variables are lost in the PCA transformation, the dimensional reduction decreases the complexity of the model while maintaining as much of the original information as possible.

Bonus Problem

Use the Sacramento data from the caret library by running *data(Sacramento)* after loading caret. This data is about housing prices in Sacramento, California. Remove the *zip* and *city* variables.

```
library(caret)
data(Sacramento)
sac_data <- select(Sacramento, -c("zip", "city"))
head(sac_data)
```

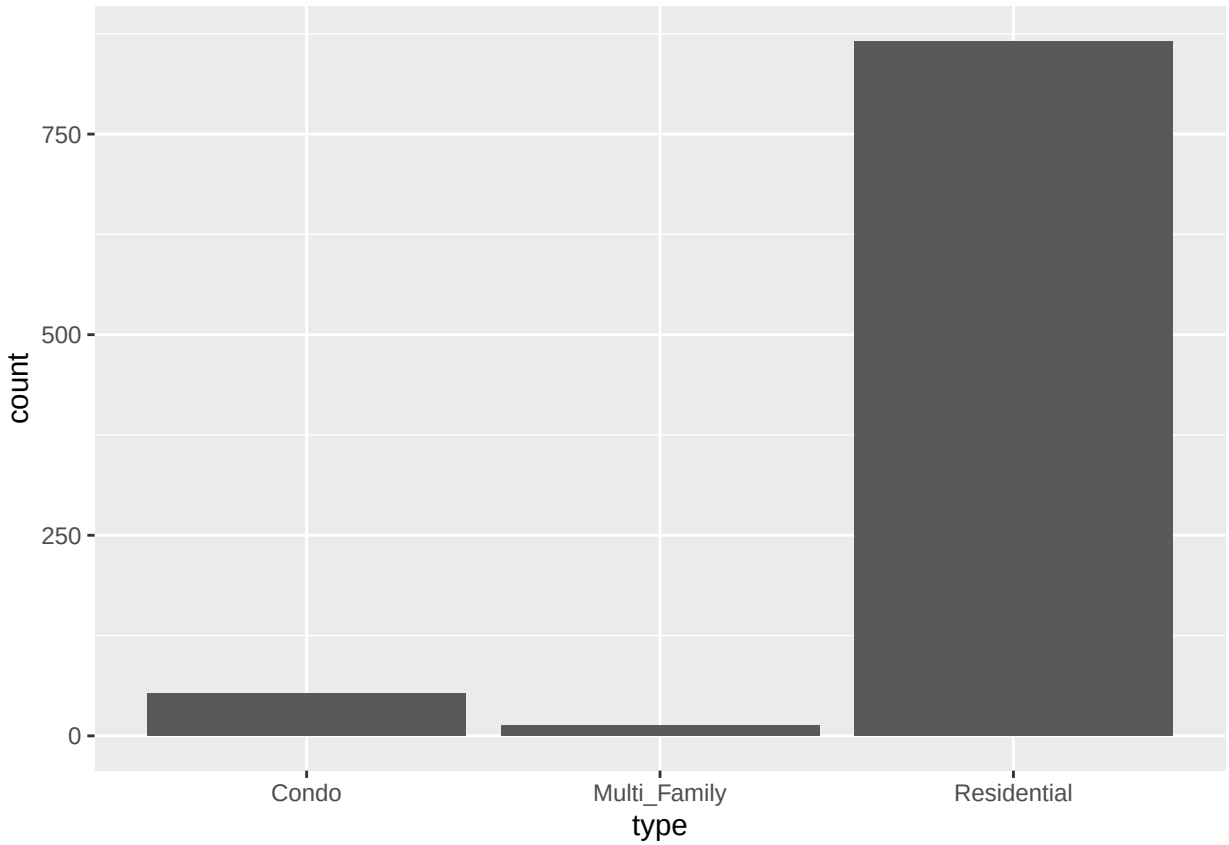
```
##   beds baths sqft        type price latitude longitude
## 1    2     1  836 Residential 59222 38.63191 -121.4349
## 2    3     1 1167 Residential 68212 38.47890 -121.4310
## 3    2     1  796 Residential 68880 38.61830 -121.4438
## 4    2     1  852 Residential 69307 38.61684 -121.4391
## 5    2     1  797 Residential 81900 38.51947 -121.4358
## 6    3     1 1122       Condo 89921 38.66260 -121.3278
```

```
summary(sac_data)
```

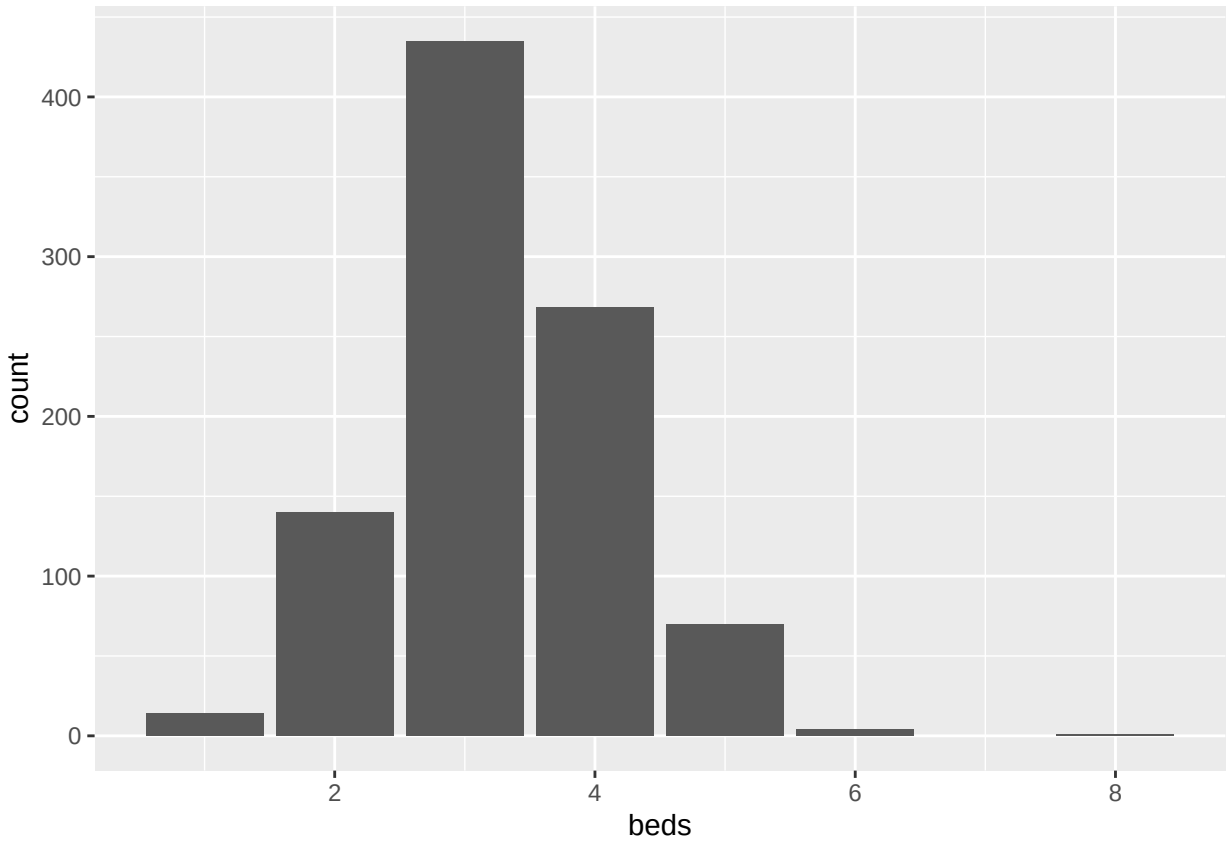
```
##       beds            baths             sqft                 type
##  Min.   :1.000   Min.   :1.000   Min.   : 484   Condo       : 53
##  1st Qu.:3.000   1st Qu.:2.000   1st Qu.:1167   Multi_Family: 13
##  Median :3.000   Median :2.000   Median :1470   Residential :866
##  Mean   :3.276   Mean   :2.053   Mean   :1680
##  3rd Qu.:4.000   3rd Qu.:2.000   3rd Qu.:1954
##  Max.   :8.000   Max.   :5.000   Max.   :4878
##      price            latitude       longitude
##  Min.   : 30000   Min.   :38.24   Min.   :-121.6
##  1st Qu.:156000   1st Qu.:38.48   1st Qu.:-121.4
##  Median :220000   Median :38.62   Median :-121.4
##  Mean   :246662   Mean   :38.59   Mean   :-121.4
##  3rd Qu.:305000   3rd Qu.:38.69   3rd Qu.:-121.3
##  Max.   :884790   Max.   :39.02   Max.   :-120.6
```

- a: Explore the variables to see if they have reasonable distributions and show your work. We will be predicting the type variable – does that mean we have a class imbalance?

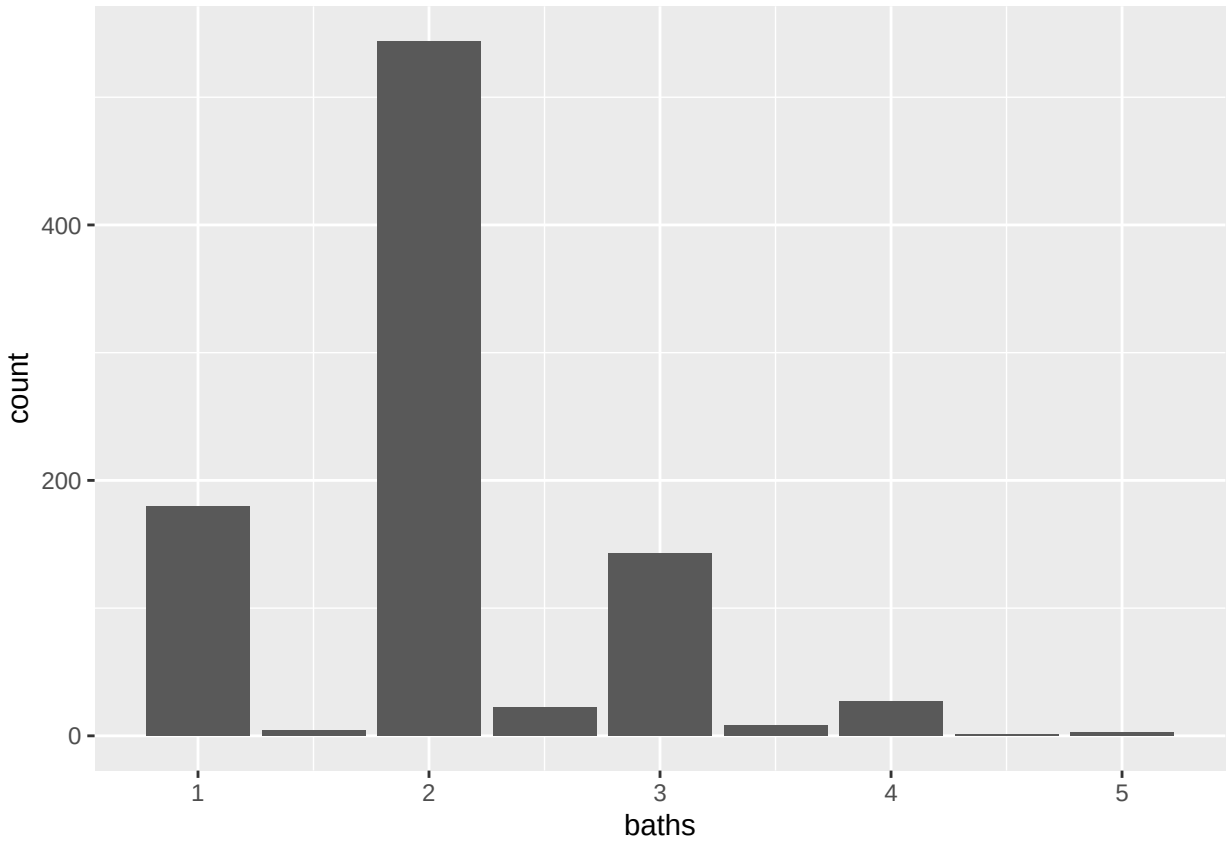
```
ggplot(sac_data, aes(type)) + geom_bar()
```



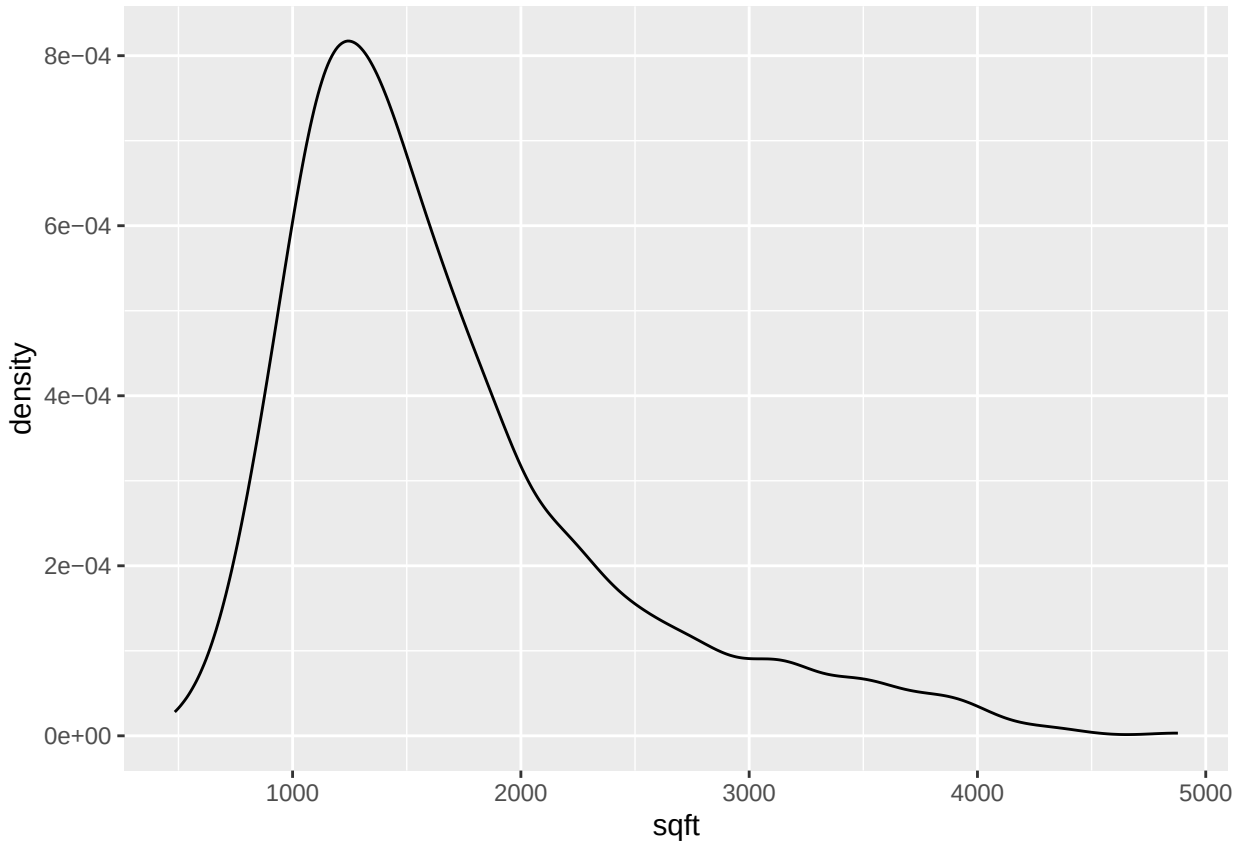

```
ggplot(sac_data, aes(beds)) + geom_bar()
```



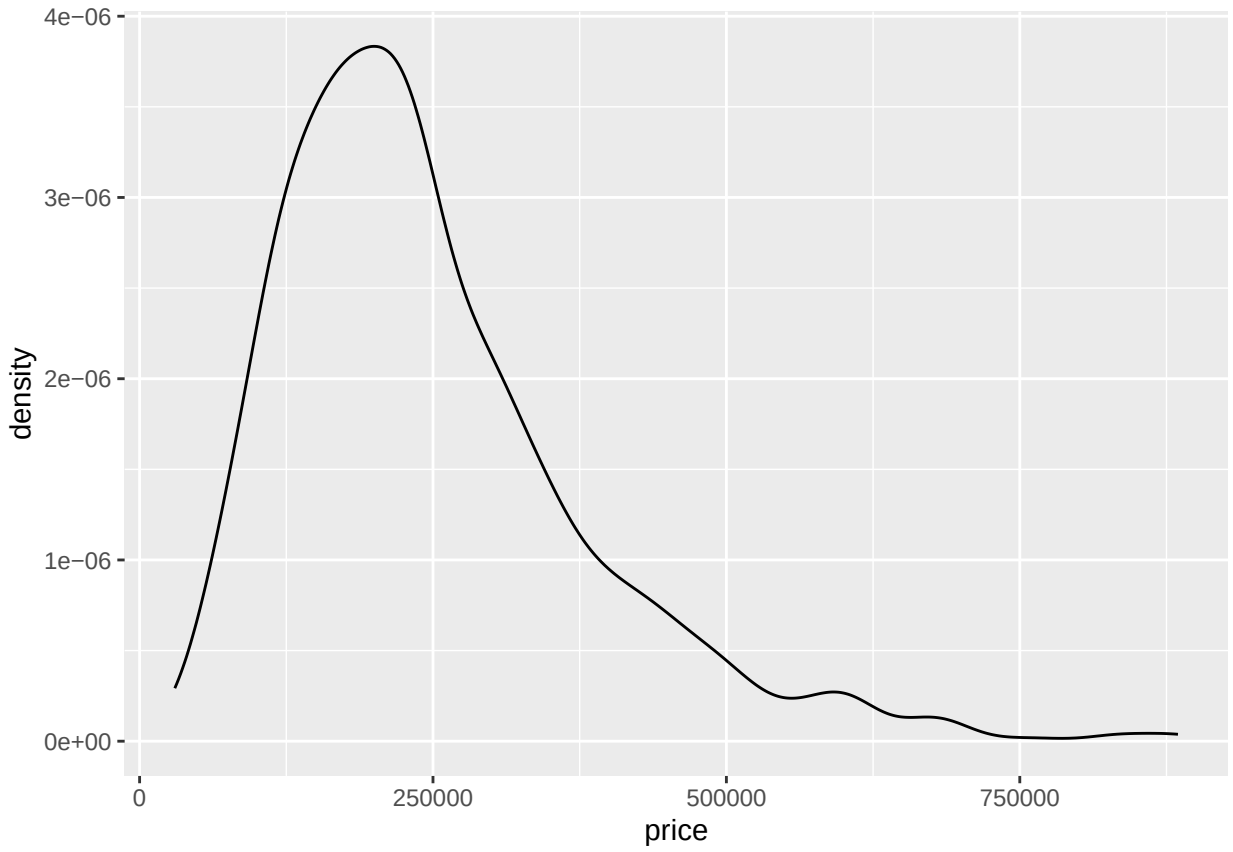

```
ggplot(sac_data, aes(baths)) + geom_bar()
```



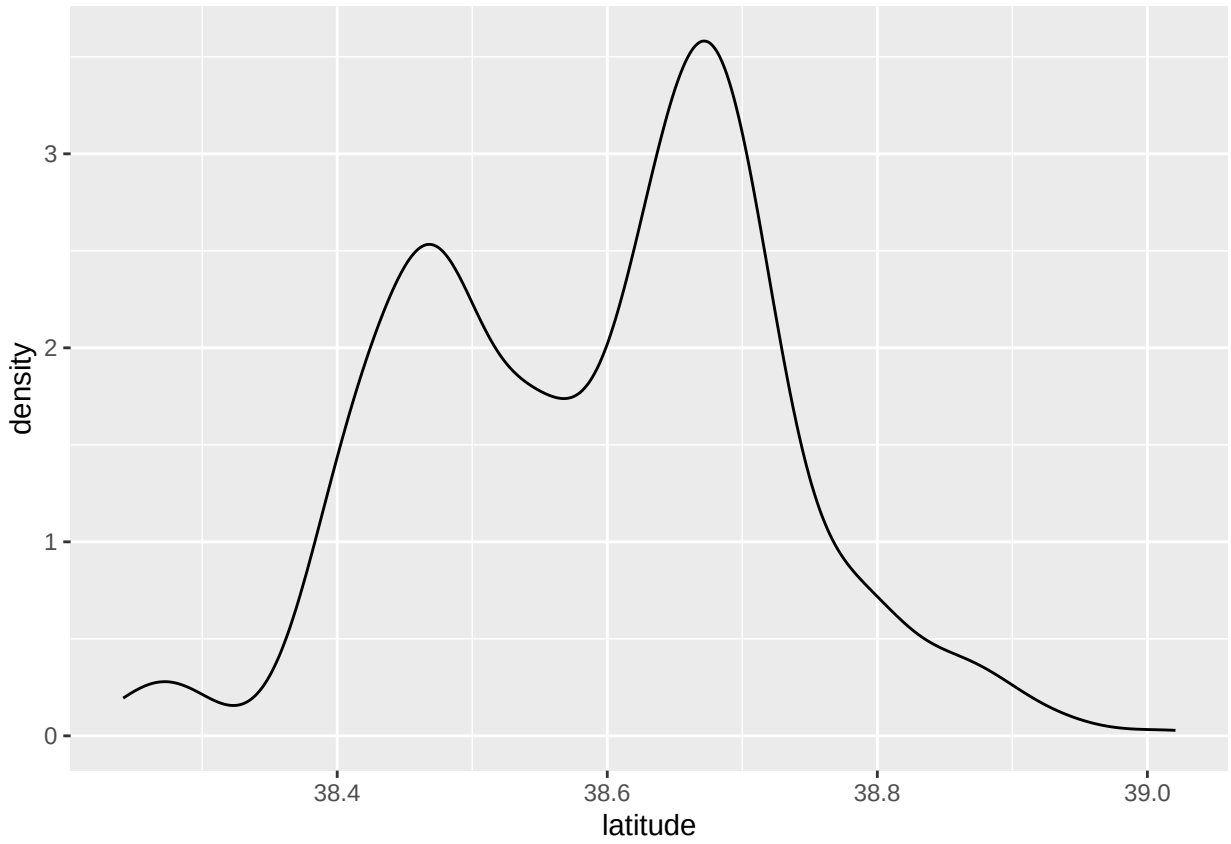

```
ggplot(sac_data, aes(sqft)) + geom_density()
```



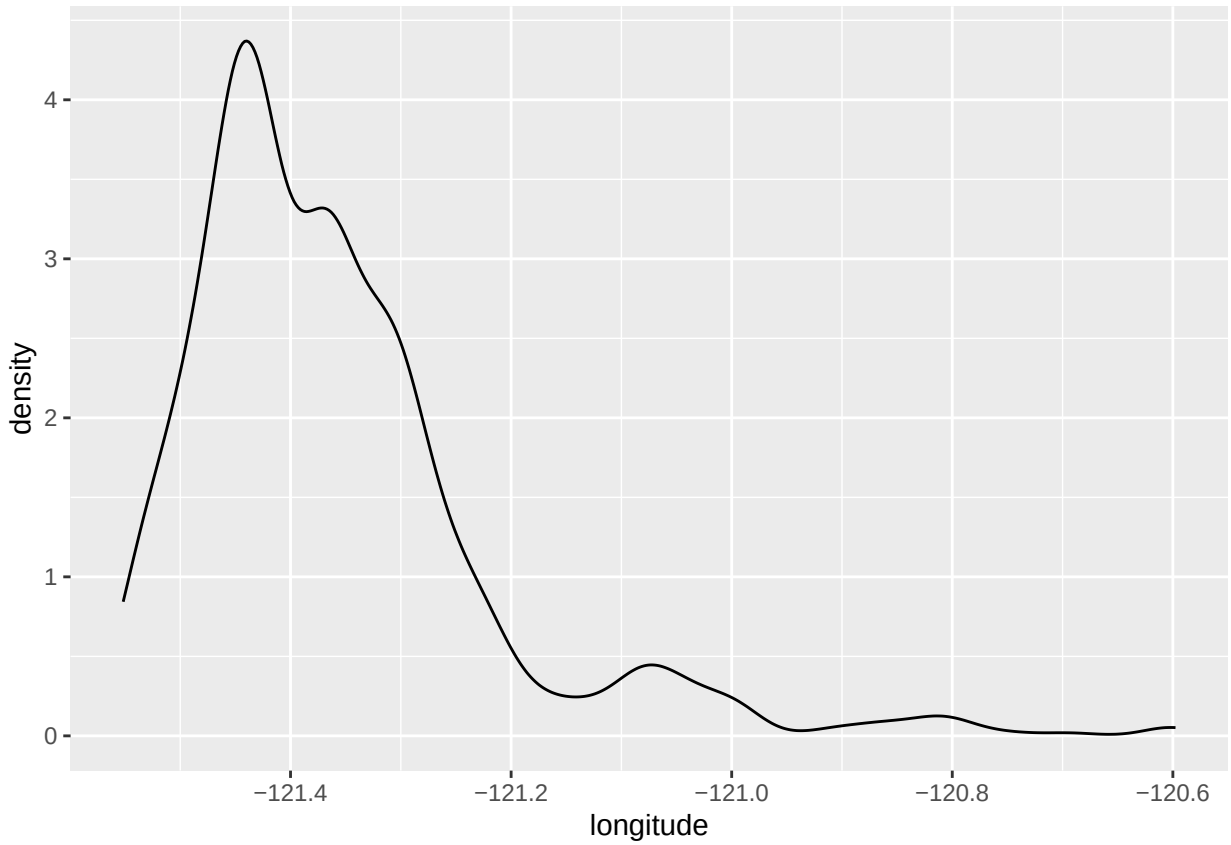

```
ggplot(sac_data, aes(price)) + geom_density()
```




```
ggplot(sac_data, aes(latitude)) + geom_density()
```




```
ggplot(sac_data, aes(longitude)) + geom_density()
```



As seen from the bar graph for the type categorical variable, there is a major imbalance in homes that are condo, multi-family, vs residential. There are significantly more residential properties than there are condos and multi-family homes. Multi-family homes have the fewest occurrences.

- b: There are lots of options for working on the data to try to improve the performance of SVM, including (1) removing other variables that you know should not be part of the prediction, (2) dealing with extreme variations in some variables with smoothing, normalization or a log transform, (3) applying PCA, and (4) to removing outliers. Pick one now and continue.

```
# remove variables that don't need to be part of the type prediction
rm_var <- select(sac_data, -c("latitude", "longitude"))
head(rm_var)
```

```
##   beds baths sqft        type price
## 1    2     1  836 Residential 59222
## 2    3     1 1167 Residential 68212
## 3    2     1  796 Residential 68880
## 4    2     1  852 Residential 69307
## 5    2     1  797 Residential 81900
## 6    3     1 1122       Condo 89921
```

- c: Use SVM to predict type and use grid search to get the best accuracy you can. The accuracy may be good, but look at the confusion matrix as well. Report what you find. Note that the kappa value provided with your SVM results can also help you see this. It is a measure of how well the classifier performed that takes into account the frequency of the classes.

```
sac_control <- trainControl(method = 'cv', number = 10)
rm_var_svm <- train(type ~., data = rm_var, method = 'svmLinear', trControl = sac_control, tuneGrid = g
rm_var_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 932 samples
##   4 predictor
##   3 classes: 'Condo', 'Multi_Family', 'Residential'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 839, 839, 839, 839, 838, 839, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.9292138   0.000000000
##   3.162278e-05  0.9292138   0.000000000
##   1.000000e-04  0.9292138   0.000000000
##   3.162278e-04  0.9292138   0.000000000
##   1.000000e-03  0.9292138   0.000000000
##   3.162278e-03  0.9292138   0.000000000
##   1.000000e-02  0.9292138   0.000000000
##   3.162278e-02  0.9292138   0.000000000
##   1.000000e-01  0.9292138   0.000000000
##   3.162278e-01  0.9281500  -0.001759134
##   1.000000e+00  0.9313758   0.091519287
##   3.162278e+00  0.9313758   0.091519287
##   1.000000e+01  0.9303005   0.089800537
##   3.162278e+01  0.9303005   0.089800537
##   1.000000e+02  0.9303005   0.089800537
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.
```

```
rm_var_svm_predict <- predict(rm_var_svm, rm_var)
head(rm_var_svm_predict)
```

```
## [1] Residential Residential Residential Residential Residential Residential
## Levels: Condo Multi_Family Residential
```

```
confusionMatrix(rm_var_svm_predict, rm_var$type)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction     Condo Multi_Family Residential
##   Condo           11            0           3
##   Multi_Family     0            0           0
##   Residential     42           13         863
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9378
##                  95% CI : (0.9203, 0.9524)
##     No Information Rate : 0.9292
##     P-Value [Acc > NIR] : 0.1694
##
##                   Kappa : 0.2584
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Condo Class: Multi_Family Class: Residential
## Sensitivity               0.20755             0.00000              0.9965
## Specificity               0.99659             1.00000              0.1667
## Pos Pred Value            0.78571                 NaN              0.9401
## Neg Pred Value            0.95425             0.98605              0.7857
## Prevalence                0.05687             0.01395              0.9292
## Detection Rate            0.01180             0.00000              0.9260
## Detection Prevalence      0.01502             0.00000              0.9850
## Balanced Accuracy         0.60207             0.50000              0.5816
```

The final value used for the model was C = 1 with accuracy: 0.9324. However, the kappa value for the parameter value of C = 1 was 0.1137. The low kappa value indicates the classifier type has 11.37% accuracy when taking into account the frequency of the predictor class, which in this case is type of home.

- d: Return to (b) and try at least one other way to try to improve the data before running SVM again, as in (c).

```
# remove classifier type variable, PCA cannot run with classifier varirable, PCA can only run with nume
pca_sac_data <- select(rm_var, -c("type"))
head(pca_sac_data)
```

```
##   beds baths sqft price
## 1    2     1  836 59222
## 2    3     1 1167 68212
## 3    2     1  796 68880
## 4    2     1  852 69307
## 5    2     1  797 81900
## 6    3     1 1122 89921
```

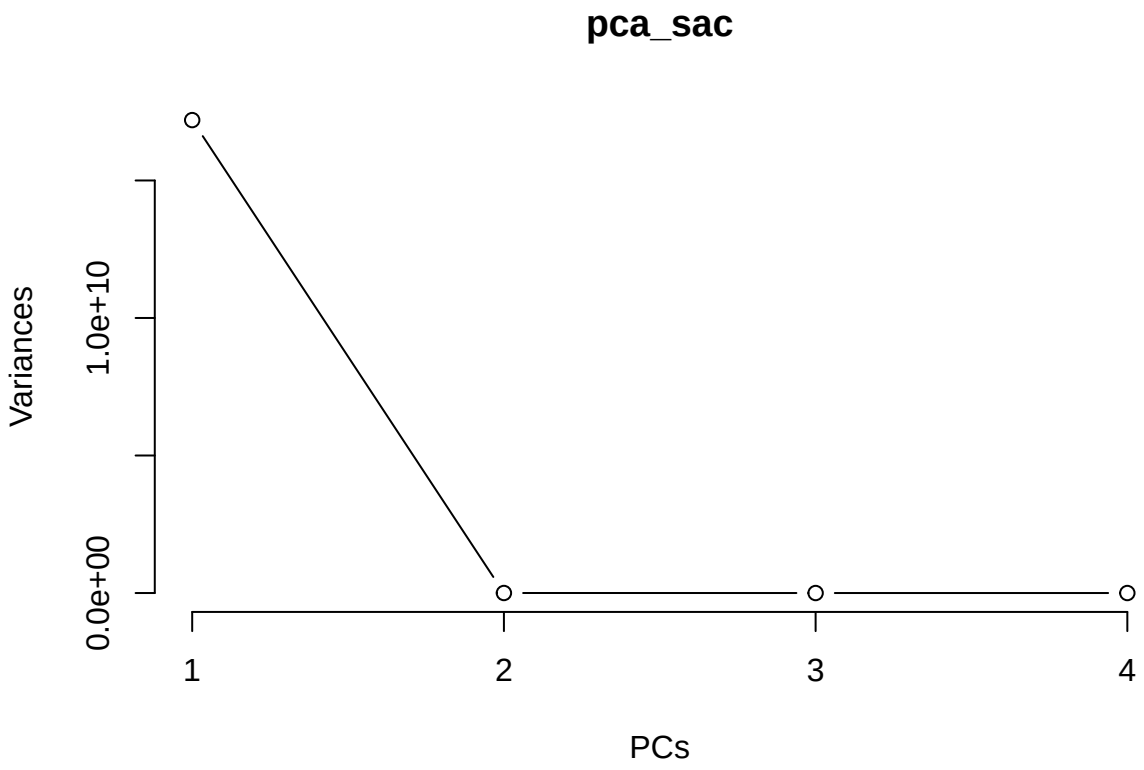
```
# remove any underrepresented variables, near zero variance attributes
nzv_sac <- nearZeroVar(pca_sac_data)
# no near zero variance issue
nzv_sac
```

```
## integer(0)
```

```
# determine optimal components with PCA w/ cumulative variance
pca_sac <- prcomp(pca_sac_data)
summary(pca_sac)
```

```
## Importance of components:
##                             PC1       PC2    PC3    PC4
## Standard deviation     131128 465.68940 0.6312 0.4379
## Proportion of Variance      1   0.00001 0.0000 0.0000
## Cumulative Proportion       1   1.00000 1.0000 1.0000
```

```
# draw screeplot
screeplot(pca_sac, type = "l") + title(xlab = "PCs")
```




```
## integer(0)
```

```
# preprocess data for PCA
sca_preProc <- preProcess(pca_sac_data, method = "pca", pcaComp = 2)
# reduce data into principal components
pca_sac_pc <- predict(sca_preProc, pca_sac_data)
# reintroduce type variable
pca_sac_pc$type <- rm_var$type
head(pca_sac_pc)
```

```
##         PC1         PC2        type
## 1 -2.724898 -0.03573814 Residential
## 2 -1.908414  0.57300920 Residential
## 3 -2.720446 -0.08360521 Residential
## 4 -2.676714 -0.09485749 Residential
## 5 -2.673052 -0.15682122 Residential
## 6 -1.864562  0.45831226       Condo
```

```r
# train svm model with pca
pca_svm_sac <- train(type ~., data = rm_var, method = "svmLinear", trControl = sac_control, tuneGrid = g
pca_svm_sac
```

```
## Support Vector Machines with Linear Kernel
##
## 932 samples
##   4 predictor
##   3 classes: 'Condo', 'Multi_Family', 'Residential'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 837, 840, 840, 839, 839, 839, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.9292332  0.0000000
##   3.162278e-05  0.9292332  0.0000000
##   1.000000e-04  0.9292332  0.0000000
##   3.162278e-04  0.9292332  0.0000000
##   1.000000e-03  0.9292332  0.0000000
##   3.162278e-03  0.9292332  0.0000000
##   1.000000e-02  0.9292332  0.0000000
##   3.162278e-02  0.9292332  0.0000000
##   1.000000e-01  0.9292332  0.0000000
##   3.162278e-01  0.9292332  0.0000000
##   1.000000e+00  0.9313607  0.1067056
##   3.162278e+00  0.9313607  0.1202150
##   1.000000e+01  0.9313607  0.1202150
##   3.162278e+01  0.9313607  0.1202150
##   1.000000e+02  0.9313607  0.1202150
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.
```

```r
pca_svm_sac_predict <- predict(pca_svm_sac, rm_var)
confusionMatrix(as.factor(pca_sac_pc$type), pca_svm_sac_predict)
```

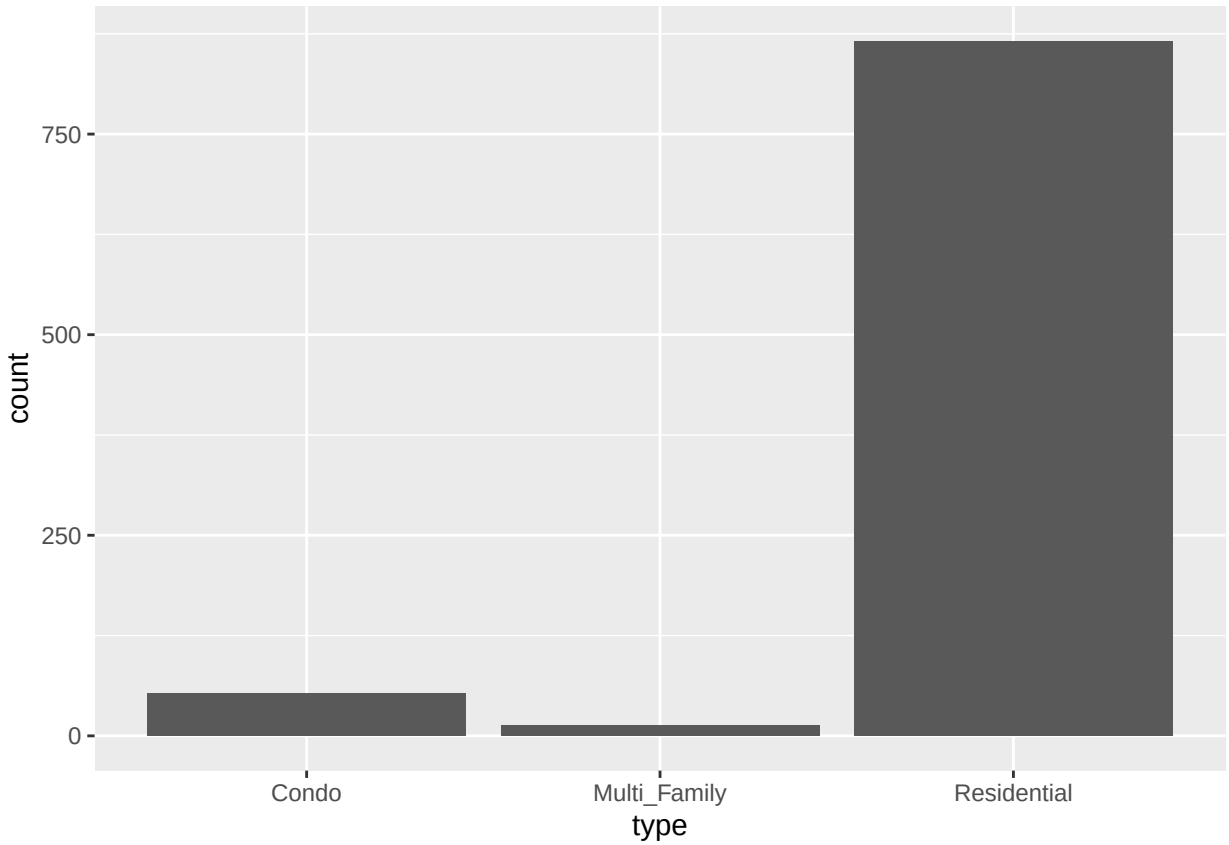
```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction     Condo Multi_Family Residential
##   Condo           11            0          42
##   Multi_Family     0            0          13
##   Residential      3            0         863
##
## Overall Statistics
##
##                Accuracy : 0.9378
##                  95% CI : (0.9203, 0.9524)
##     No Information Rate : 0.985
##     P-Value [Acc > NIR] : 1
```

```
##
##                   Kappa : 0.2584
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Condo Class: Multi_Family Class: Residential
## Sensitivity               0.78571                  NA             0.9401
## Specificity               0.95425             0.98605             0.7857
## Pos Pred Value            0.20755                  NA             0.9965
## Neg Pred Value            0.99659                  NA             0.1667
## Prevalence                0.01502             0.00000             0.9850
## Detection Rate            0.01180             0.00000             0.9260
## Detection Prevalence      0.05687             0.01395             0.9292
## Balanced Accuracy         0.86998                  NA             0.8629
```

- e: In the end, some data are just so imbalanced that a classifier is never going to predict the minority class. Dealing with this is a huge topic. One simple possibility is to conclude that we do not have enough data to support predicting the very infrequent class(es) and remove them. If they are not actually important to the reason we are making the prediction, that could be fine. Another approach is to force the data to be more even by sampling.

Create a copy of the data that includes all the data from the two smaller classes, plus a small random sample of the large class (you can do this by separating those data with a filter, sampling, then attaching them back on). Check the distributions of the variables in this new data sample to make sure they are reasonably close to the originals using visualization and/or summary statistics. We want to make sure we did not get a strange sample where everything was cheap or there were only studio apartments, for example. You can rerun the sampling a few times if you are getting strange results. If it keeps happening, check your process.

```
ggplot(sac_data, aes(type)) + geom_bar()
```




```
summary(sac_data$type)
```

```
##        Condo Multi_Family  Residential
##           53           13          866
```

```
multi <- filter(sac_data, type == "Multi_Family")
condo <- filter(sac_data, type == "Condo")
total_residential <- filter(sac_data, type == "Residential")
residential <- total_residential[sample(nrow(total_residential), size = 100), ]
head(residential)
```

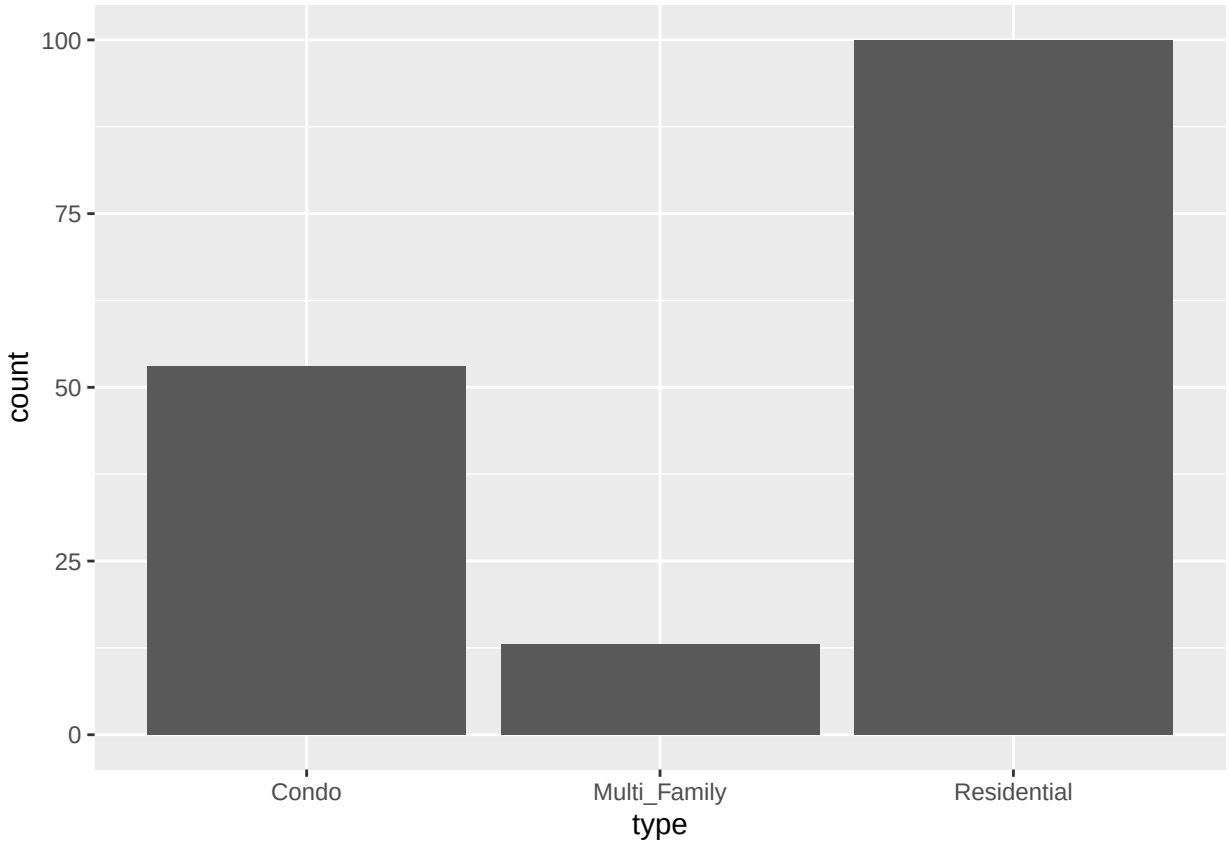
```
##     beds baths sqft        type  price latitude longitude
## 509    2     1  838 Residential  55422 38.47165 -121.4352
## 87     5     3 2790 Residential 258000 38.42568 -121.4381
## 649    3     2 1540 Residential 266510 38.38771 -121.4365
## 707    3     2 2960 Residential 350000 38.70951 -121.3594
## 345    2     1 1139 Residential 133105 38.52094 -121.4594
## 217    3     2 1522 Residential 225000 38.62476 -121.5228
```

```
#append database into one
new_data <- rbind(multi, condo, residential)
summary(new_data)
```

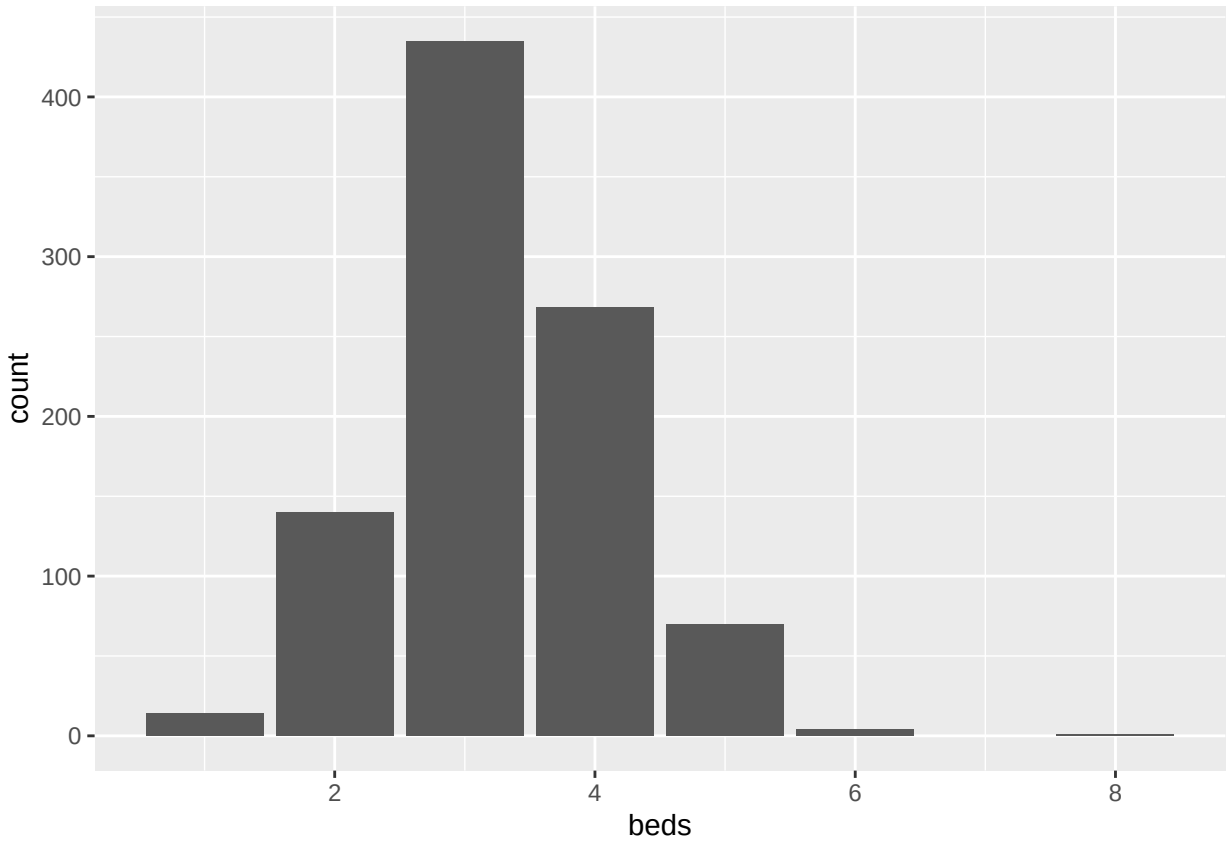
```
##       beds           baths            sqft                type
##  Min.   :1.000   Min.   :1.000   Min.   : 484   Condo       : 53
```

```
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1039   Multi_Family: 13
##  Median :3.000   Median :2.000   Median :1307   Residential :100
##  Mean   :2.922   Mean   :1.934   Mean   :1497
##  3rd Qu.:4.000   3rd Qu.:2.000   3rd Qu.:1743
##  Max.   :8.000   Max.   :4.000   Max.   :3714
##      price           latitude       longitude
##  Min.   : 40000   Min.   :38.25   Min.   :-121.5
##  1st Qu.:126970   1st Qu.:38.49   1st Qu.:-121.4
##  Median :191250   Median :38.62   Median :-121.4
##  Mean   :209535   Mean   :38.60   Mean   :-121.4
##  3rd Qu.:257514   3rd Qu.:38.67   3rd Qu.:-121.3
##  Max.   :668365   Max.   :38.94   Max.   :-120.6
```

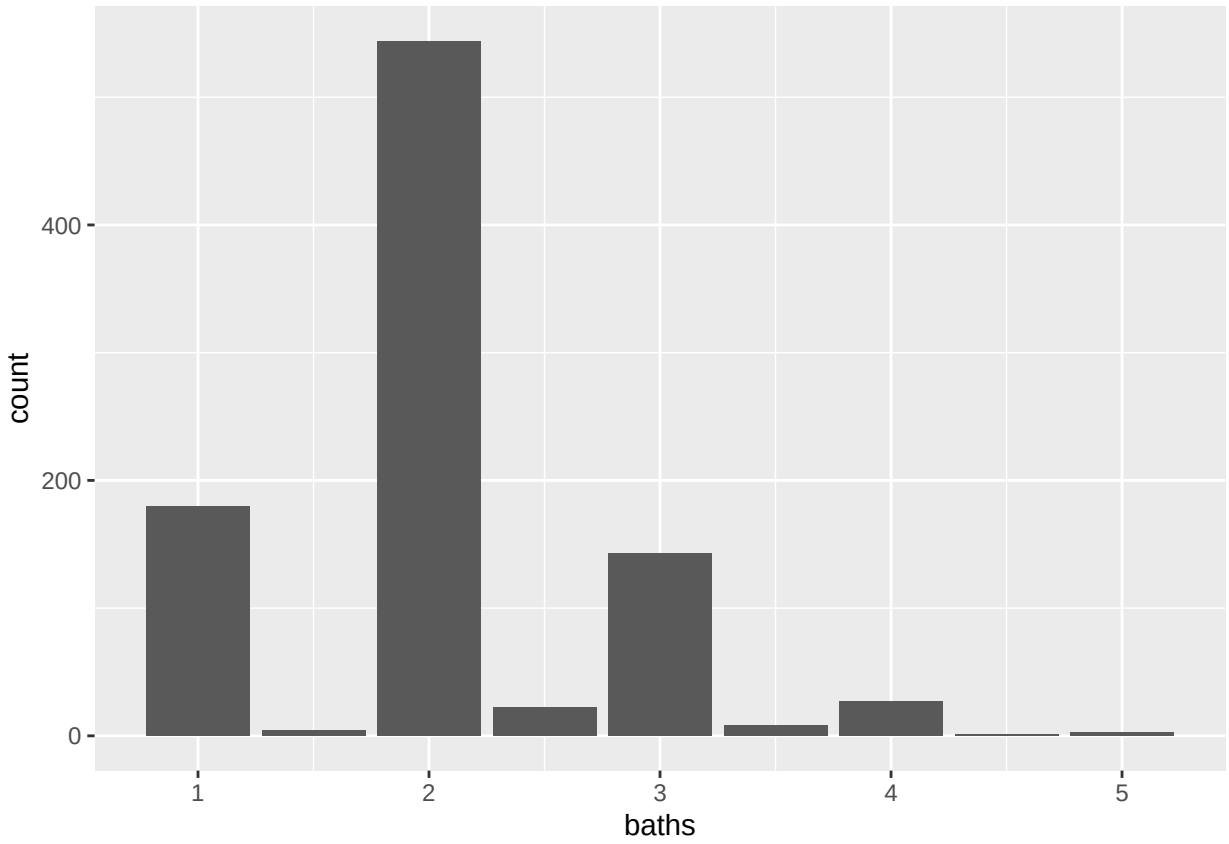
```
ggplot(new_data, aes(type)) + geom_bar()
```



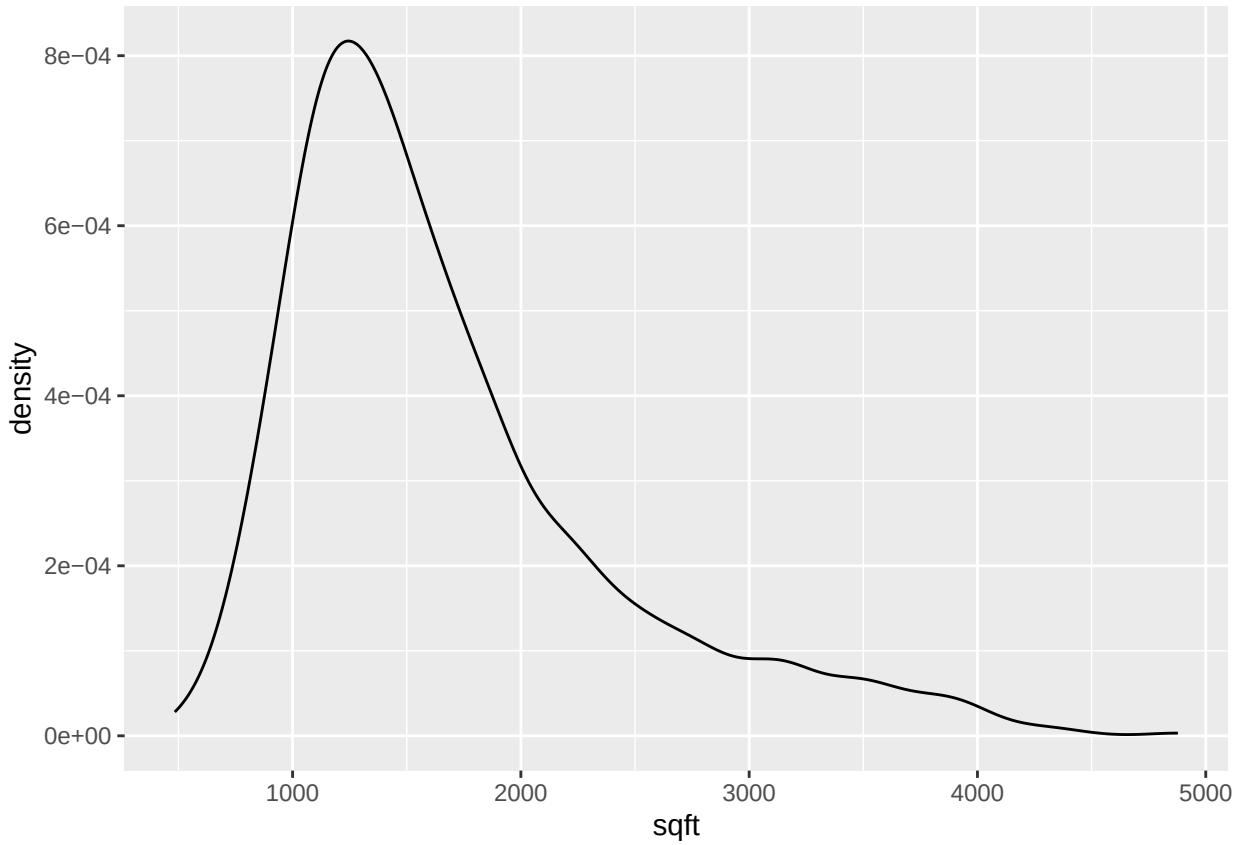

```
ggplot(sac_data, aes(beds)) + geom_bar()
```



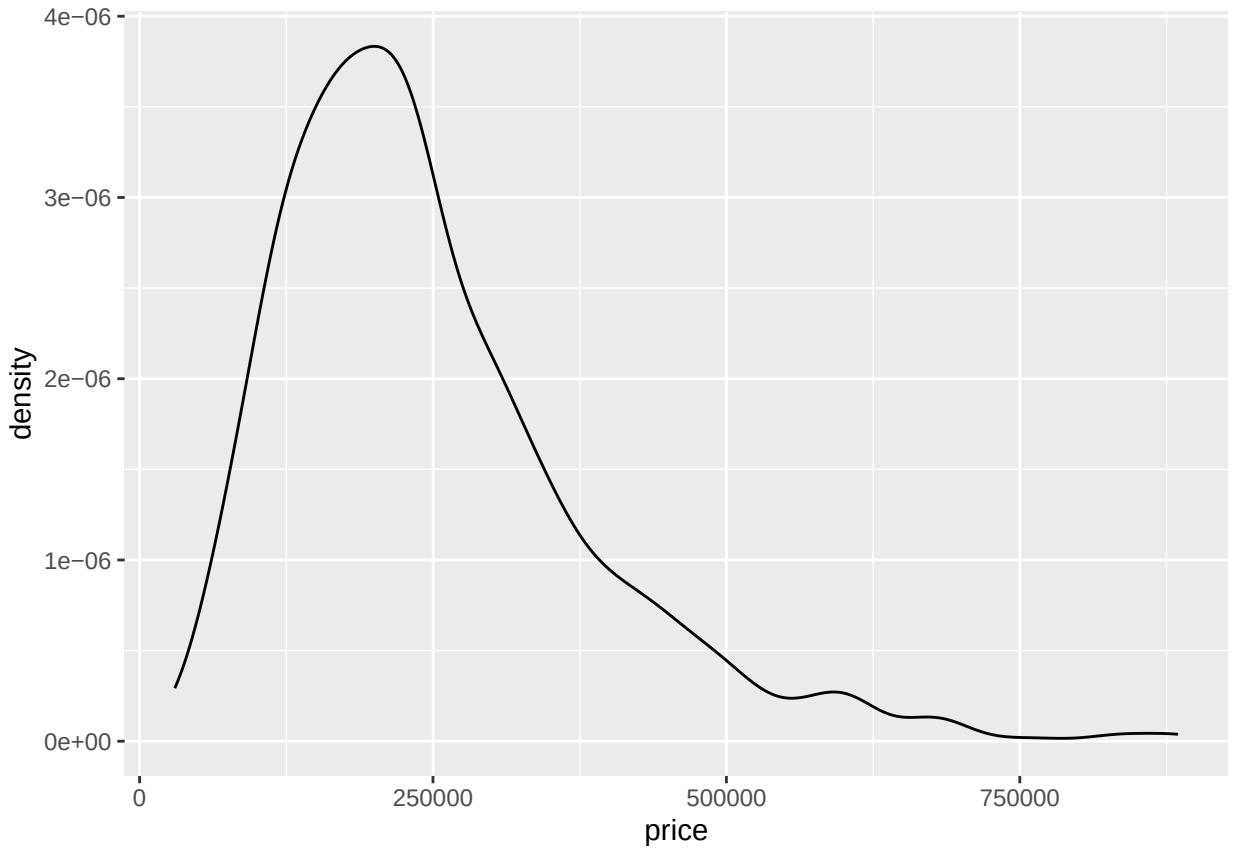

```
ggplot(sac_data, aes(baths)) + geom_bar()
```



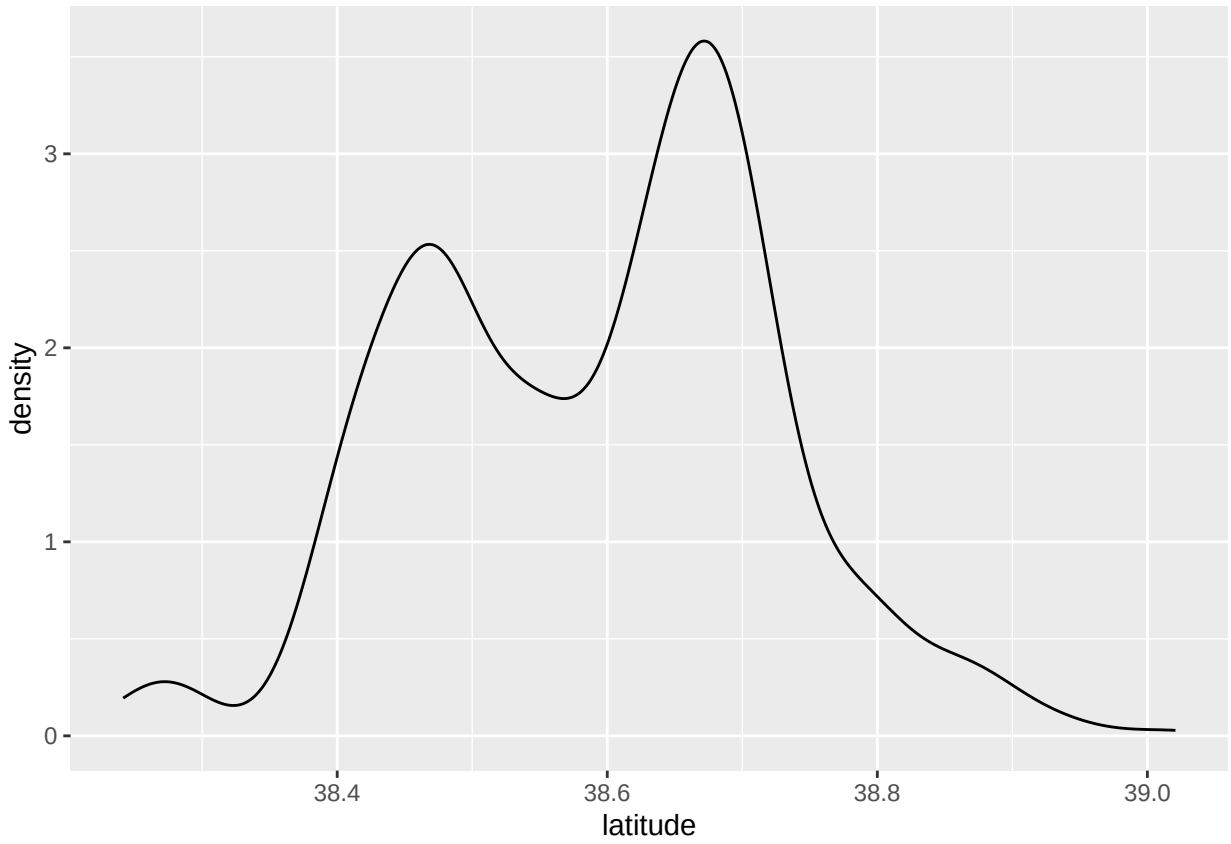

```
ggplot(sac_data, aes(sqft)) + geom_density()
```



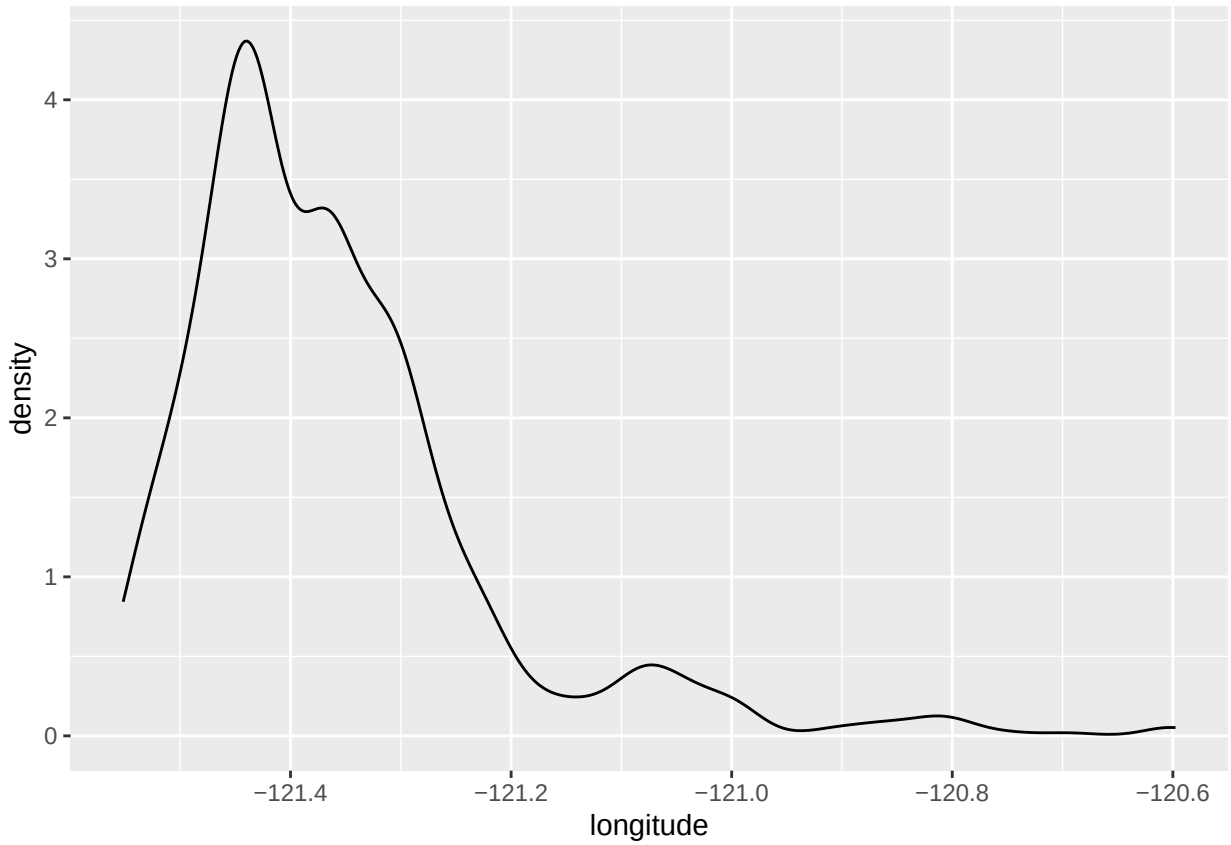

```
ggplot(sac_data, aes(price)) + geom_density()
```




```
ggplot(sac_data, aes(latitude)) + geom_density()
```




```
ggplot(sac_data, aes(longitude)) + geom_density()
```



Use SVM to predict type one this new, more balanced dataset and report its performance with a confusion matrix and with grid search to get the best accuracy.

```
new_train = trainControl(method = "cv", number = 10)
new_grid = expand.grid(C = 10^seq(-5,2,0.5))

new_svm <- train(type ~., data = new_data, method = "svmLinear", trControl = new_train, tuneGrid = new_g
new_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 166 samples
##   6 predictor
##   3 classes: 'Condo', 'Multi_Family', 'Residential'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 150, 150, 150, 148, 149, 150, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.6037582  0.0000000
##   3.162278e-05  0.6037582  0.0000000
##   1.000000e-04  0.6037582  0.0000000
##   3.162278e-04  0.6037582  0.0000000
##   1.000000e-03  0.6037582  0.0000000
##   3.162278e-03  0.6037582  0.0000000
```

```
##   1.000000e-02  0.6766340  0.2218115
##   3.162278e-02  0.7724265  0.5248802
##   1.000000e-01  0.7790441  0.5475886
##   3.162278e-01  0.8213644  0.6497600
##   1.000000e+00  0.8338644  0.6816149
##   3.162278e+00  0.8269199  0.6631730
##   1.000000e+01  0.8147876  0.6394189
##   3.162278e+01  0.8147876  0.6394189
##   1.000000e+02  0.8147876  0.6394189
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.
```

```
new_predict <- predict(new_svm, new_data)
head(new_predict)
```

```
## [1] Residential  Multi_Family Multi_Family Multi_Family Multi_Family
## [6] Residential
## Levels: Condo Multi_Family Residential
```

```
confusionMatrix(new_data$type, new_predict)
```

```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction     Condo Multi_Family Residential
##   Condo           46            0           7
##   Multi_Family     1            7           5
##   Residential     13            0          87
##
## Overall Statistics
##
##                Accuracy : 0.8434
##                  95% CI : (0.779, 0.8951)
##     No Information Rate : 0.5964
##     P-Value [Acc > NIR] : 4.832e-12
##
##                   Kappa : 0.7
##
##  Mcnemar's Test P-Value : 0.05033
##
## Statistics by Class:
##
##                      Class: Condo Class: Multi_Family Class: Residential
## Sensitivity                0.7667             1.00000             0.8788
## Specificity                0.9340             0.96226             0.8060
## Pos Pred Value             0.8679             0.53846             0.8700
## Neg Pred Value             0.8761             1.00000             0.8182
## Prevalence                 0.3614             0.04217             0.5964
## Detection Rate             0.2771             0.04217             0.5241
## Detection Prevalence       0.3193             0.07831             0.6024
## Balanced Accuracy          0.8503             0.98113             0.8424
```

The final value used for the model was C = 0.316227 with accuracy of 0.8382 and kappa of 0.6851. As a result of the confusion matrix, the accuracy of the predicted values was 0.8434 with a kappa value of 0.7. With a newly balanced classifier, the accuracy of the model dropped, however, the kappa value increased substantially. The trade-off in accuracy for increased kappa value is well worth the overall generalization while taking into consideration the frequency of the classifier classes.

Bonus Problem

To understand just how much different subsets can differ, create a 5 fold partitioning of the cars data included in R *(mtcars)* and visualize the distribution of the *gears* variable across the folds. Rather than use the fancy *trainControl* methods for making the folds, create them directly so you actually can keep track of which data points are in which fold. This is not covered in the tutorial, but it is quick. Here is code to create 5 folds and a variable in the data frame that contains the fold index of each point. Use that resulting data frame to create your visualization.

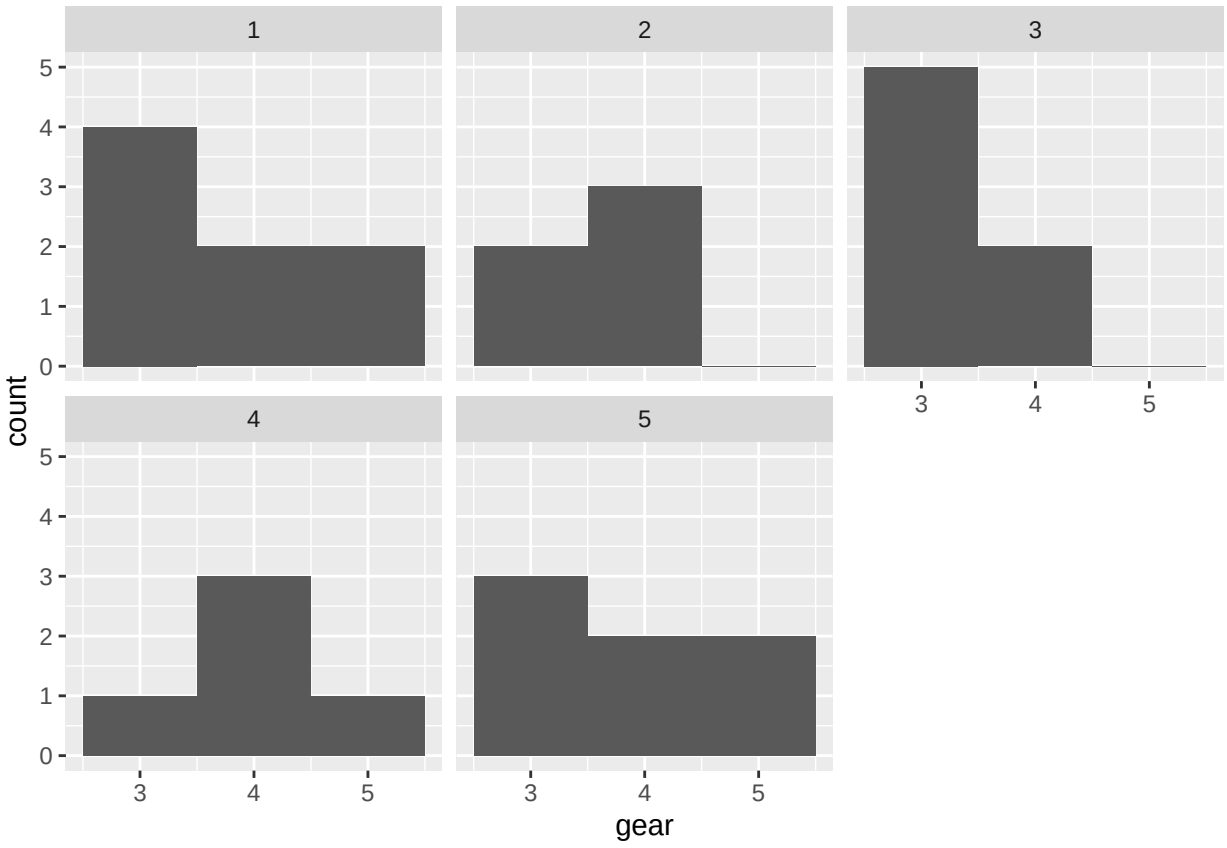
```
mycars <- mtcars # make a copy to modify
mycars$folds = 0 # initialize new variable to hold fold indices
head(mycars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb folds
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4     0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4     0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1     0
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1     0
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2     0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1     0
```

```
# Create 5 folds, get a list of lists of indices.
# Take a look at this result so you understand what is happening.
# Note we are not passing the data frame directly, but a list of its indices created by 1:nrow(mycars).
flds = createFolds(1:nrow(mycars), k=5, list=TRUE)
# This loop sets all the rows in a given fold to have that fold's index in the folds variable. Take a l
for (i in 1:5) { mycars$folds[flds[[i]]] = i}
head(mycars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb folds
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4     5
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4     1
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1     2
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1     3
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2     1
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1     4
```

```
ggplot(mycars, aes(gear)) + geom_histogram(binwidth = 1) + facet_wrap(~folds)
```


1
2
3
4
5
count
gear
0
1
2
3
4
5