Assignment_2

BARINI SIMHADRI

Importing required libraries.

```
library(caret)
library(ggplot2)
library(e1071)
library(tidyverse)
```

Importing bank data set.

```
bank_data = read.csv("C:/Users/bunty/Desktop/funda/week_4/BankData.csv",header = T, sep=",")
```

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.

checking for the class of each columns. here, bool1 , bool2, bool3, approval are categorical and we will convert it into numerical.

```
str(bank_data)
```

```
## 'data.frame':    690 obs. of  13 variables:
##  $ X         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ cont1     : num  30.8 58.7 24.5 27.8 20.2 ...
##  $ cont2     : num  0 4.46 0.5 1.54 5.62 ...
##  $ cont3     : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ bool1     : chr  "t" "t" "t" "t" ...
```

```
## $ bool2        : chr  "t" "t" "f" "t" ...
## $ cont4        : int  1 6 0 5 0 0 0 0 0 0 ...
## $ bool3        : chr  "f" "f" "f" "t" ...
## $ cont5        : int  202 43 280 100 120 360 164 80 180 52 ...
## $ cont6        : int  0 560 824 3 0 0 31285 1349 314 1442 ...
## $ approval     : chr  "+" "+" "+" "+" ...
## $ credit.score: num  665 694 622 654 670 ...
## $ ages         : int  42 54 29 58 65 61 50 41 30 35 ...
```

There are NA's present in some of the columns.

```
summary(bank_data)
```

```
##        X             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
##  Length:690         Length:690         Min.   : 0.0   Length:690
##  Class :character   Class :character   1st Qu.: 0.0   Class :character
##  Mode  :character   Mode  :character   Median : 0.0   Mode  :character
##                                        Mean   : 2.4
##                                        3rd Qu.: 3.0
##                                        Max.   :67.0
##
##      cont5          cont6             approval          credit.score
##  Min.   :   0   Min.   :     0.0   Length:690         Min.   :583.7
##  1st Qu.:  75   1st Qu.:     0.0   Class :character   1st Qu.:666.7
##  Median : 160   Median :     5.0   Mode  :character   Median :697.3
##  Mean   : 184   Mean   :  1017.4                      Mean   :696.4
##  3rd Qu.: 276   3rd Qu.:   395.5                      3rd Qu.:726.4
##  Max.   :2000   Max.   :100000.0                      Max.   :806.0
##  NA's   :13
##       ages
##  Min.   :11.00
##  1st Qu.:31.00
##  Median :38.00
##  Mean   :39.67
##  3rd Qu.:48.00
##  Max.   :84.00
##
```

checking all the rows with NA's, and there are total 24 rows with NA's.

```
bank_data[rowSums(is.na(bank_data)) > 0, ]
```

```
##       X cont1  cont2  cont3 bool1 bool2 cont4 bool3 cont5 cont6 approval
## 72   72 34.83  4.000 12.500     t     f     0     t    NA     0        -
```

```
## 84  84    NA  3.500 3.000   t   f   0   t  300     0   -
## 87  87    NA  0.375 0.875   t   f   0   t  928     0   -
## 93  93    NA  5.000 8.500   t   f   0   f    0     0   -
## 98  98    NA  0.500 0.835   t   f   0   t  320     0   -
## 203 203 24.83  2.750 2.250   t   t   6   f   NA   600   +
## 207 207 71.58  0.000 0.000   f   f   0   f   NA     0   +
## 244 244 18.75  7.500 2.710   t   t   5   f   NA 26726   +
## 255 255    NA  0.625 0.250   f   f   0   f  380  2010   -
## 271 271 37.58  0.000 0.000   f   f   0   f   NA     0   +
## 279 279 24.58 13.500 0.000   f   f   0   f   NA     0   -
## 287 287    NA  1.500 0.000   f   t   2   t  200   105   -
## 330 330    NA  4.000 0.085   f   f   0   t  411     0   -
## 331 331 20.42  0.000 0.000   f   f   0   f   NA     0   -
## 407 407 40.33  8.125 0.165   f   t   2   f   NA    18   -
## 446 446    NA 11.250 0.000   f   f   0   f   NA  5200   -
## 451 451    NA  3.000 7.000   f   f   0   f    0     1   -
## 457 457 34.58  0.000 0.000   f   f   0   f   NA     0   -
## 501 501    NA  4.000 5.000   t   t   3   t  290  2279   +
## 516 516    NA 10.500 6.500   t   f   0   f    0     0   +
## 593 593 23.17  0.000 0.000   f   f   0   f   NA     0   +
## 609 609    NA  0.040 4.250   f   f   0   t  460     0   -
## 623 623 25.58  0.000 0.000   f   f   0   f   NA     0   +
## 627 627 22.00  7.835 0.165   f   f   0   t   NA     0   -
##     credit.score ages
## 72        674.26   53
## 84        752.21   38
## 87        677.58   30
## 93        699.88   53
## 98        723.07   60
## 203       729.35   65
## 207       647.30   40
## 244       685.64   31
## 255       703.11   32
## 271       721.43   32
## 279       778.61   33
## 287       706.52   36
## 330       653.67   29
## 331       630.27   32
## 407       682.31   29
## 446       748.25   32
## 451       754.04   32
## 457       680.82   27
## 501       701.82   40
## 516       673.82   41
## 593       681.89   47
## 609       787.79   49
## 623       703.16   36
## 627       583.66   36
```

dropping all the rows with NA's and we can see that all the rows with NA's are dropped.

```
bank_data <- na.omit(bank_data)
bank_data[rowSums(is.na(bank_data)) > 0, ]
```

```
## [1] X            cont1        cont2        cont3        bool1
## [6] bool2        cont4        bool3        cont5        cont6
## [11] approval    credit.score ages
## <0 rows> (or 0-length row.names)
```

replacing categorical value to numerical.

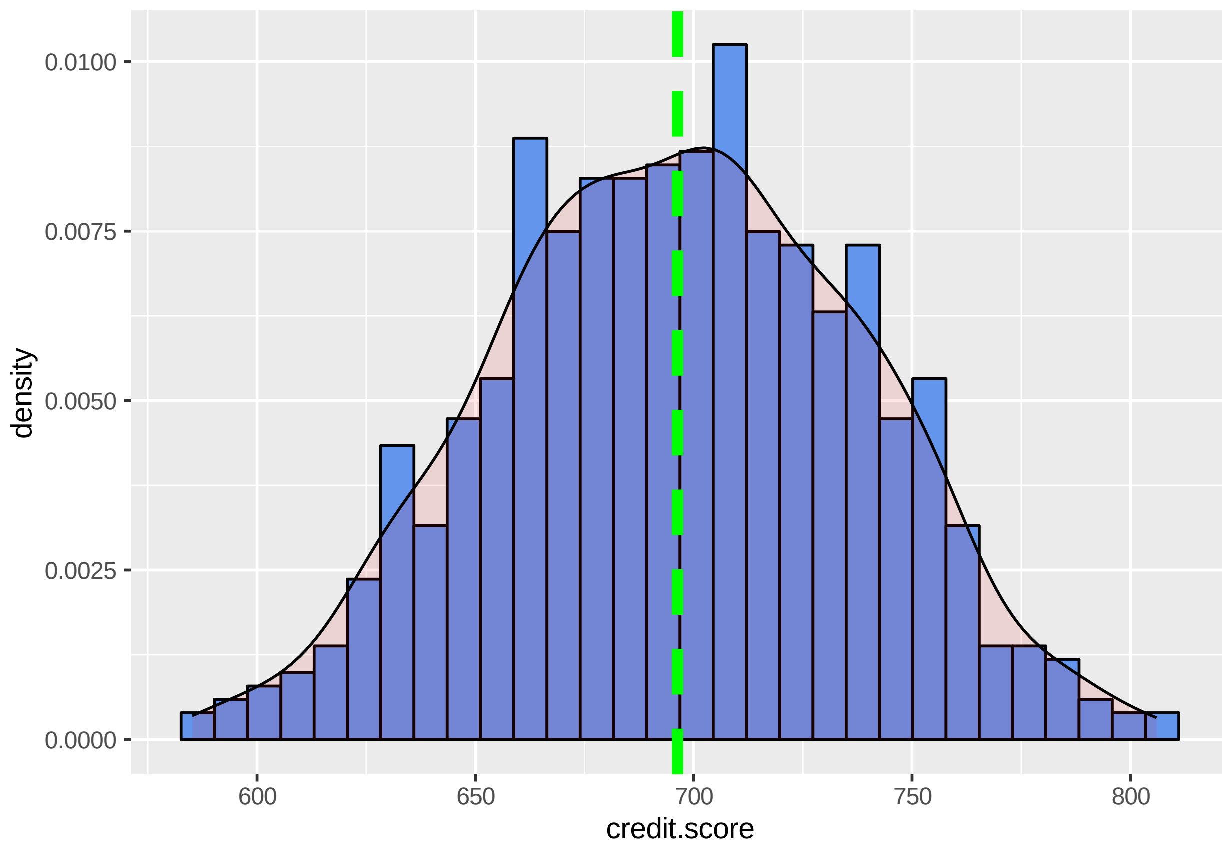
```
bank_data$bool1 <- ifelse(bank_data$bool1 == "t",1,0)
bank_data$bool2 <- ifelse(bank_data$bool2 == "t",1,0)
bank_data$bool3 <- ifelse(bank_data$bool3 == "t",1,0)
bank_data$approval <- ifelse(bank_data$approval == "+",1,0)
```

```
str(bank_data)
```

```
## 'data.frame':     666 obs. of  13 variables:
##  $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ cont1       : num  30.8 58.7 24.5 27.8 20.2 ...
##  $ cont2       : num  0 4.46 0.5 1.54 5.62 ...
##  $ cont3       : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ bool1       : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ bool2       : num  1 1 0 1 0 0 0 0 0 0 ...
##  $ cont4       : int  1 6 0 5 0 0 0 0 0 0 ...
##  $ bool3       : num  0 0 0 1 0 1 1 0 0 1 ...
##  $ cont5       : int  202 43 280 100 120 360 164 80 180 52 ...
##  $ cont6       : int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ approval    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ credit.score: num  665 694 622 654 670 ...
##  $ ages        : int  42 54 29 58 65 61 50 41 30 35 ...
##  - attr(*, "na.action")= 'omit' Named int [1:24] 72 84 87 93 98 203 207 244 255 271 ...
##   ..- attr(*, "names")= chr [1:24] "72" "84" "87" "93" ...
```

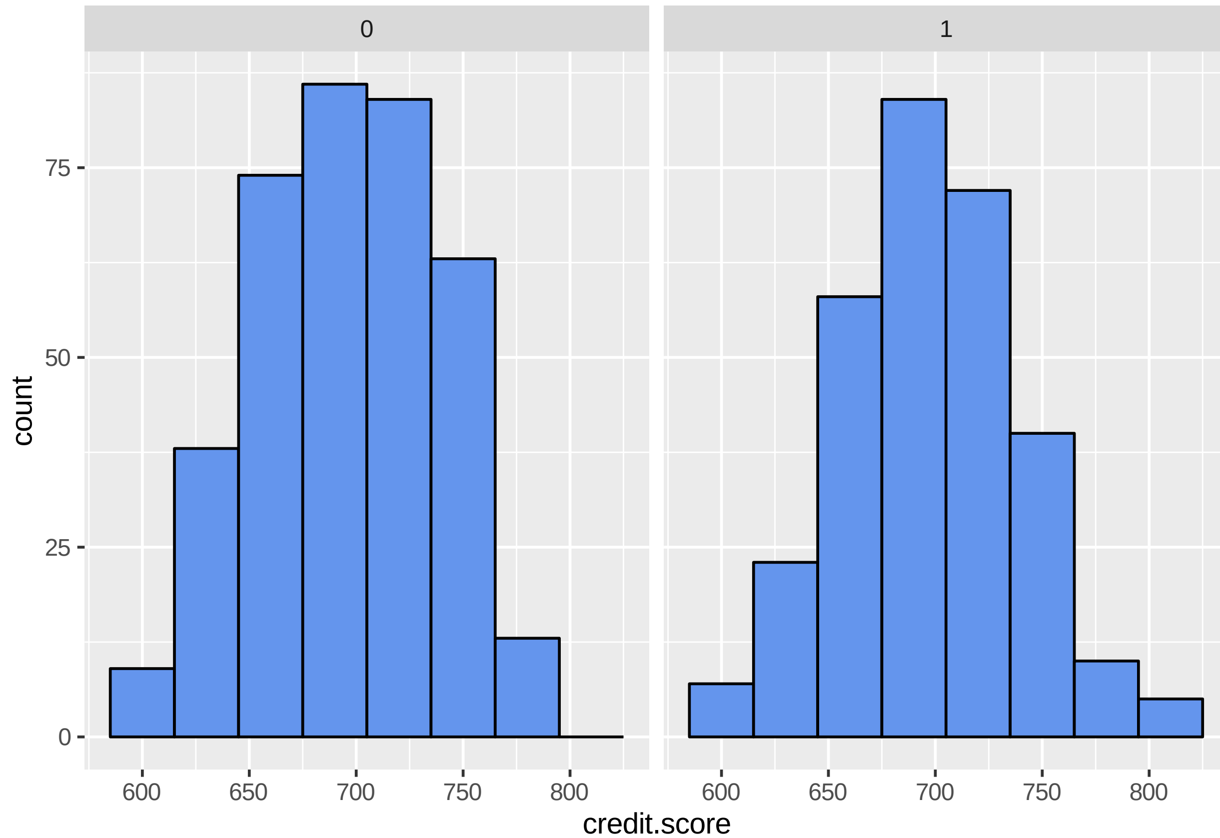
Using histogram and density plot to visualize the distribution of credit.score. It is nearly symmetrical distribution.

```
ggplot(bank_data,aes(x=credit.score)) +geom_histogram(aes(y = after_stat(density)) ,color = "black" ,fi
  geom_vline(aes(xintercept=mean(credit.score)),color = "green" ,linetype = "dashed" , linewidth = 2)
```



Distribution of credit.score wrt approval.

```
ggplot(bank_data, aes(x = credit.score)) + geom_histogram(binwidth = 30,color="black" , fill="cornflowe
facet_wrap(~approval)
```



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 normalization to column cont1. We use z-score when the data is nearly to normally distributed so that it gets normalize to the mean to 0 and standard deviation of 1 and to know how the data points are deviated from the mean of distribution.

```
summary(bank_data$cont1)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    13.75    22.60    28.50    31.57    38.25    80.25
```

```
z_score <- bank_data$cont1
z_score <- as.data.frame(z_score)
```

method = center , scale for z_score normalization

```
z_score_preproc <- preProcess(z_score,method = c("center","scale"))
standardization <- predict(z_score_preproc,z_score)
```

```
z_score$z_cont1 <- standardization
summary(z_score$z_cont1)
```

```
##         z_score
## Min. :-1.4949  ##
1st  Qu.:-0.7522
##   Median  :-0.2575
## Mean : 0.0000  ##
3rd  Qu.: 0.5605  ##
Max.     :  4.0839
```

```
z_score$z_cont1 <- unlist(z_score$z_cont1)
```

Min-Max normalization to column cont2, it rescales the value to a specific range and it is useful when we want to compare the variables that are measured in different units.

```
min_max <- bank_data$cont2
min_max <- as.data.frame(min_max)
```

```
min_max_prepoc <- preProcess(min_max,method=c("range"))
min_max_normalize<- predict(min_max_prepoc,min_max)
```

```
min_max$min_max_cont2<-min_max_normalize
summary(min_max$min_max_cont2)
```

```
##         min_max
##  Min.   :0.00000  ##
1st  Qu.:0.03607
##   Median :0.09821
## Mean   :0.17136  ##
3rd   Qu.:0.25741    ##
Max.     :1.00000
```

```
min_max$min_max_cont2<-unlist(min_max$min_max_cont2)
str(min_max)
```

```
## 'data.frame':       666  obs.  of    2  variables:
##    $ min_max        :  num  0  4.46  0.5  1.54  5.62  ...
##    $ min_max_cont2:  num  0  0.1593  0.0179  0.055  0.2009  ...
```

Decimal Scaling to cont3, it is used when we want to preserve the magnitude and scale to a common range. Also when we re dealing with large or small number in the data set.

```
decimal_scale <- bank_data$cont3
decimal_scale<- as.data.frame(decimal_scale)
```

```
decimal_scale$decimal_scale_cont3<-  decimal_scale/100
```

```
decimal_scale$decimal_scale_cont3<-  unlist(decimal_scale$decimal_scale_cont3)
```

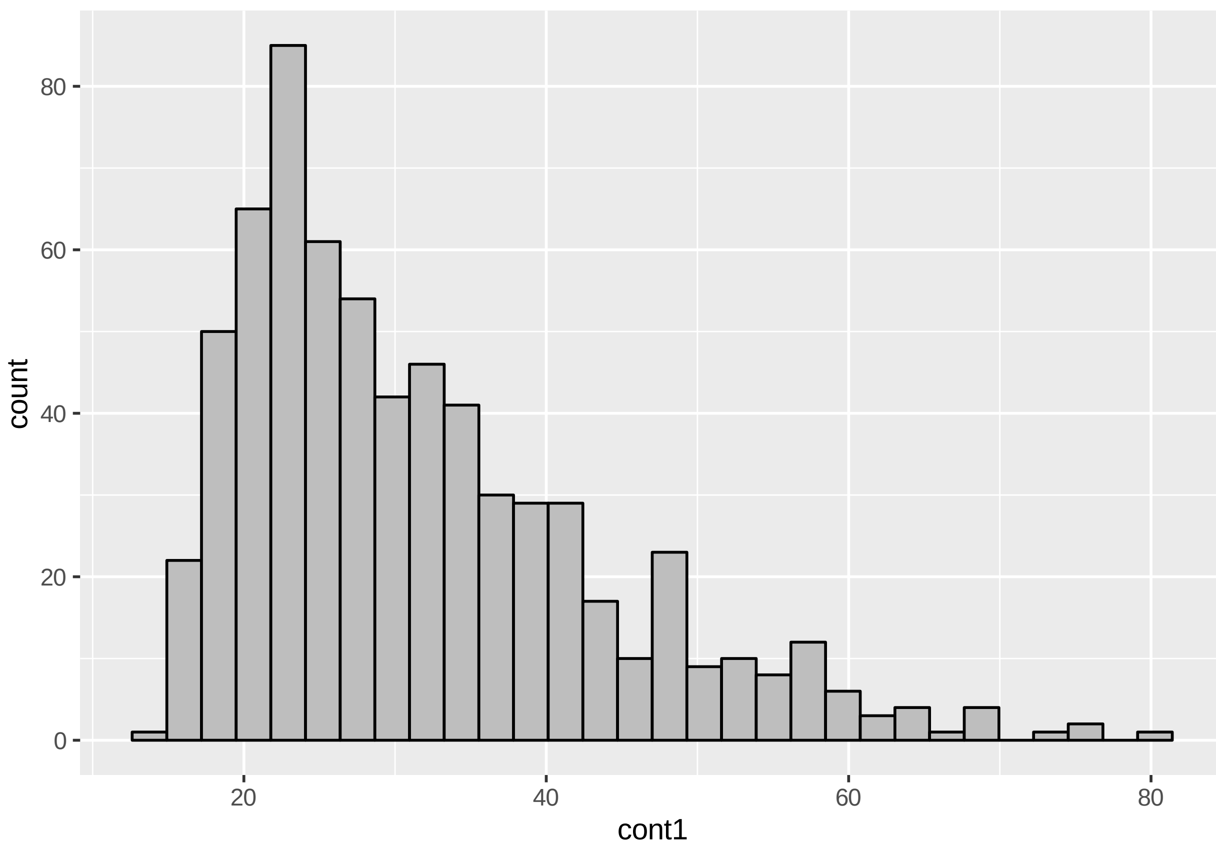
```
head(decimal_scale)
```

```
##     decimal_scale  decimal_scale_cont3
## 1            1.25               0.0125
## 2            3.04               0.0304
## 3            1.50               0.0150
## 4            3.75               0.0375
## 5            1.71               0.0171
## 6            2.50               0.0250
```

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

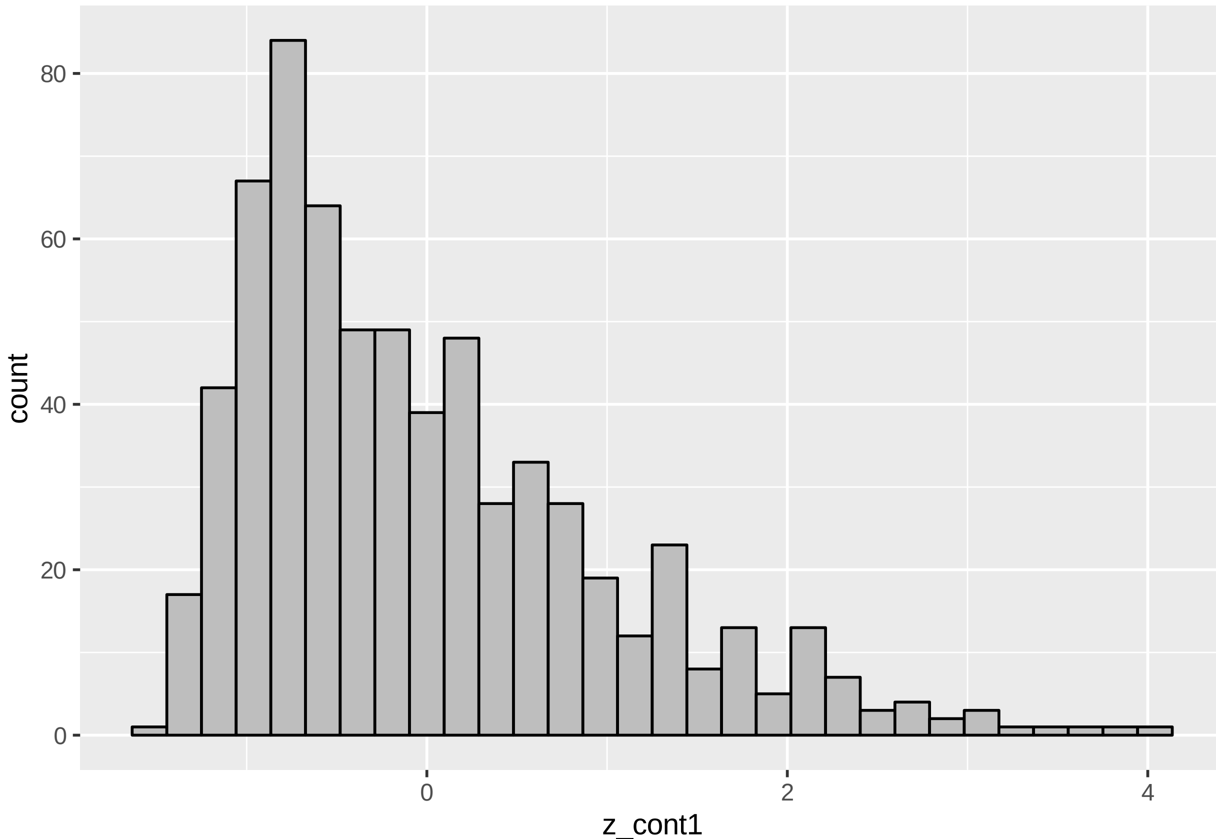
z_score normalization cont1 before

```
ggplot(bank_data,aes(x=cont1))+geom_histogram(color="black",fill="grey")
```



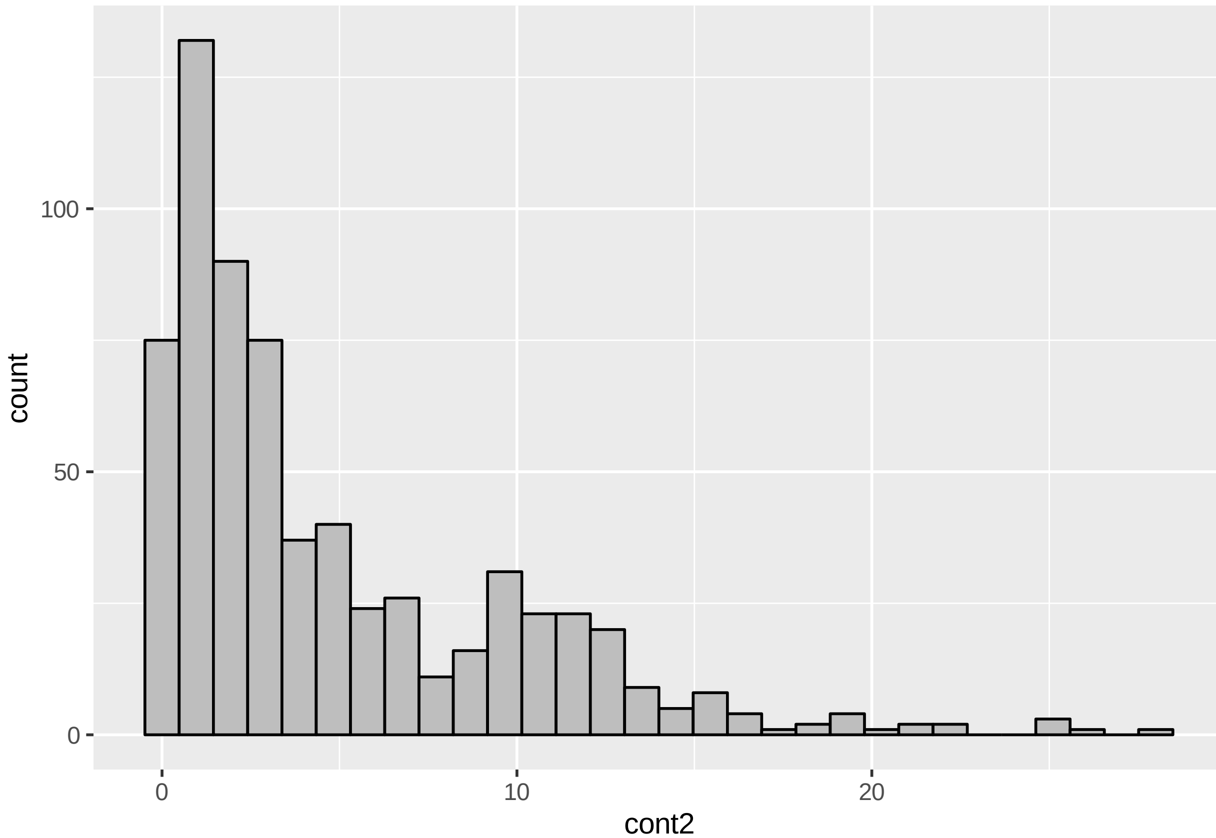
z_score normalization cont1 after

```
ggplot(z_score,aes(x=z_cont1))+geom_histogram(color="black",fill="grey")
```



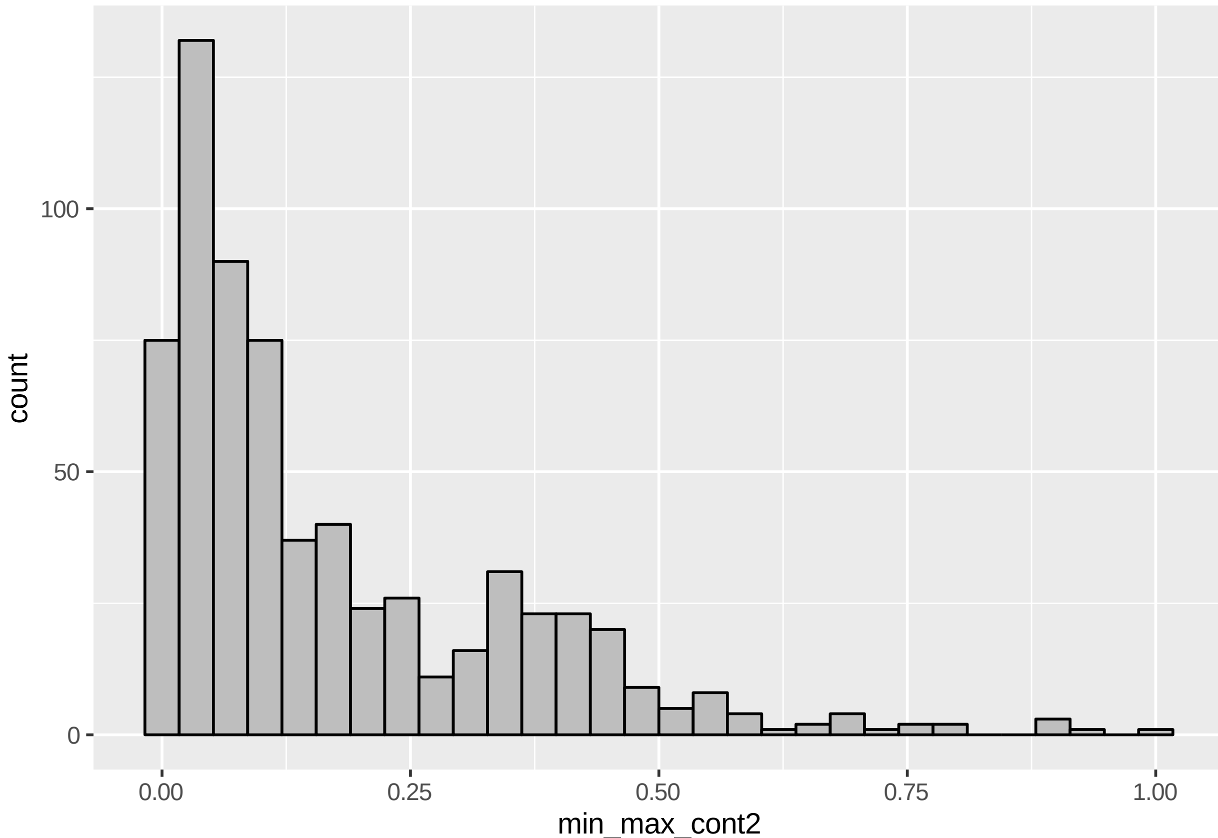
Min-Max normalization cont2 before.

```
ggplot(bank_data,aes(x=cont2))+geom_histogram(color="black",fill="grey")
```



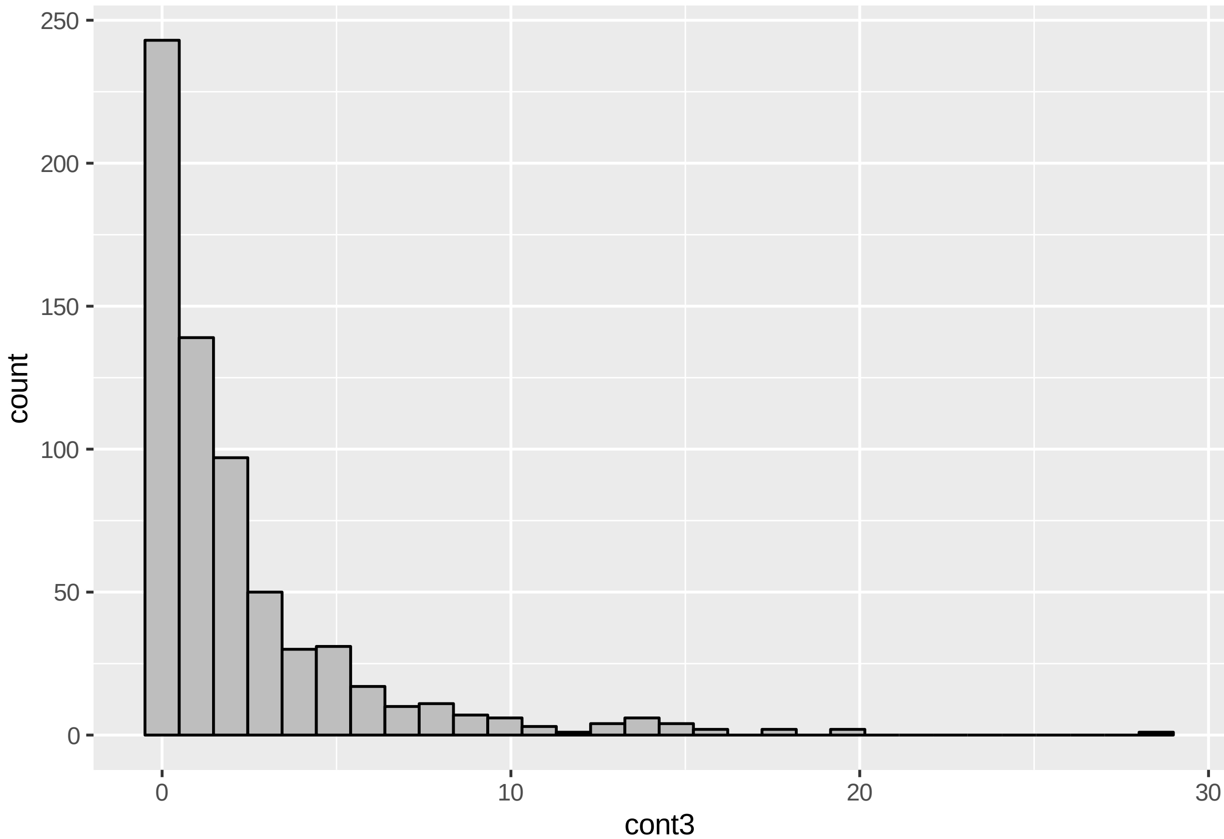
Min-Max normalization cont2 after.

```
ggplot(min_max,aes(x=min_max_cont2))+geom_histogram(color="black",fill="grey")
```



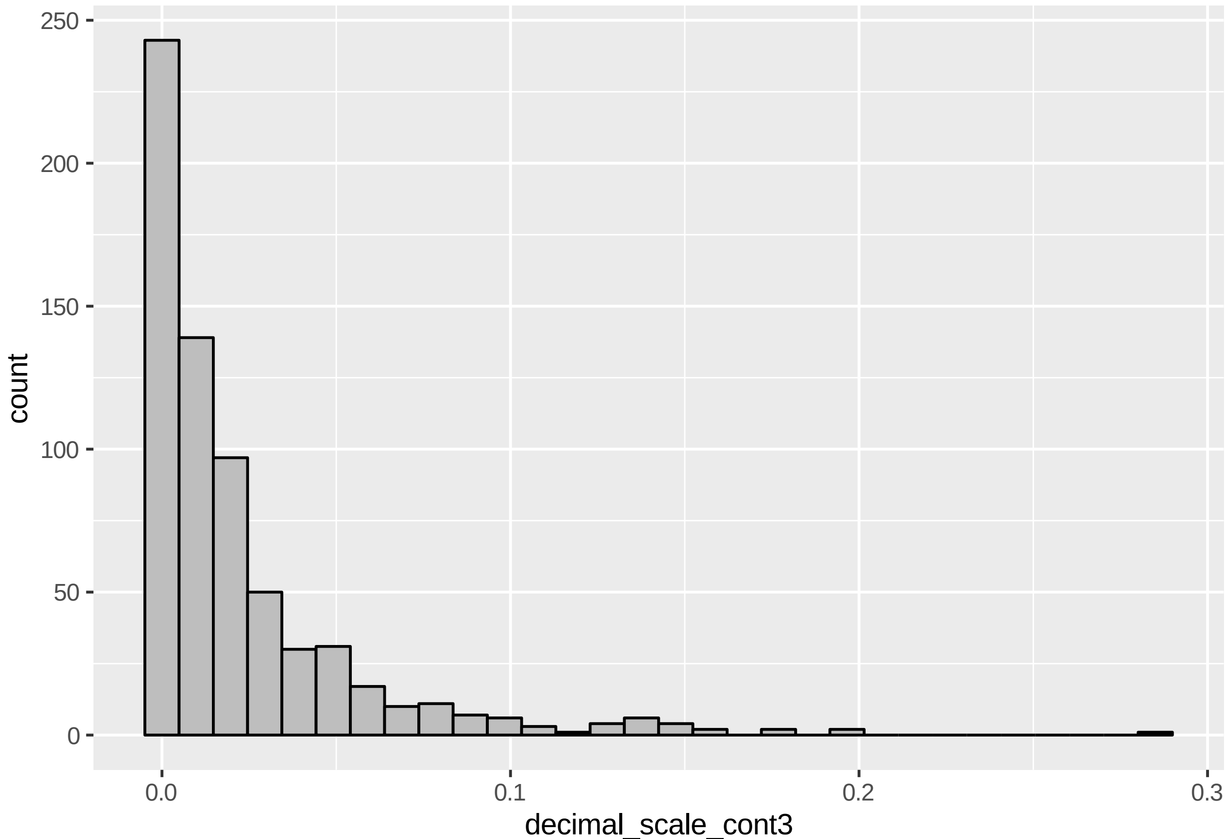
decimal-scaling normalization cont3 before.

```
ggplot(bank_data,aes(x=cont3))+geom_histogram(color="black",fill="grey")
```



Min-Max normalization cont3 after.

```
ggplot(decimal_scale,aes(x=decimal_scale_cont3))+geom_histogram(color="black",fill="grey")
```



we can observer the visualization of before and after normalization and conclude that there is no change in the distribution which means we have preserved the distribution and still reduced the form of a variables to get better analytics.

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?

```
IQR(bank_data$cont5)
```

```
## [1] 195.75
```

```
quantiles <- quantile(bank_data$cont5, c(0.25, 0.5, 0.75))
quantiles
```

```
##     25%    50%    75%
##   75.25 160.00 271.00
```

Here, we can observe the quantile range of cont5. We can bin them in 4 range, low -> medium-low -> medium-high -> high (custom ranges)

```
bins <- c(-Inf, quantiles[1], quantiles[2], quantiles[3], Inf)
bins
```

```
##             25%    50%    75%
##   -Inf  75.25 160.00 271.00    Inf
```

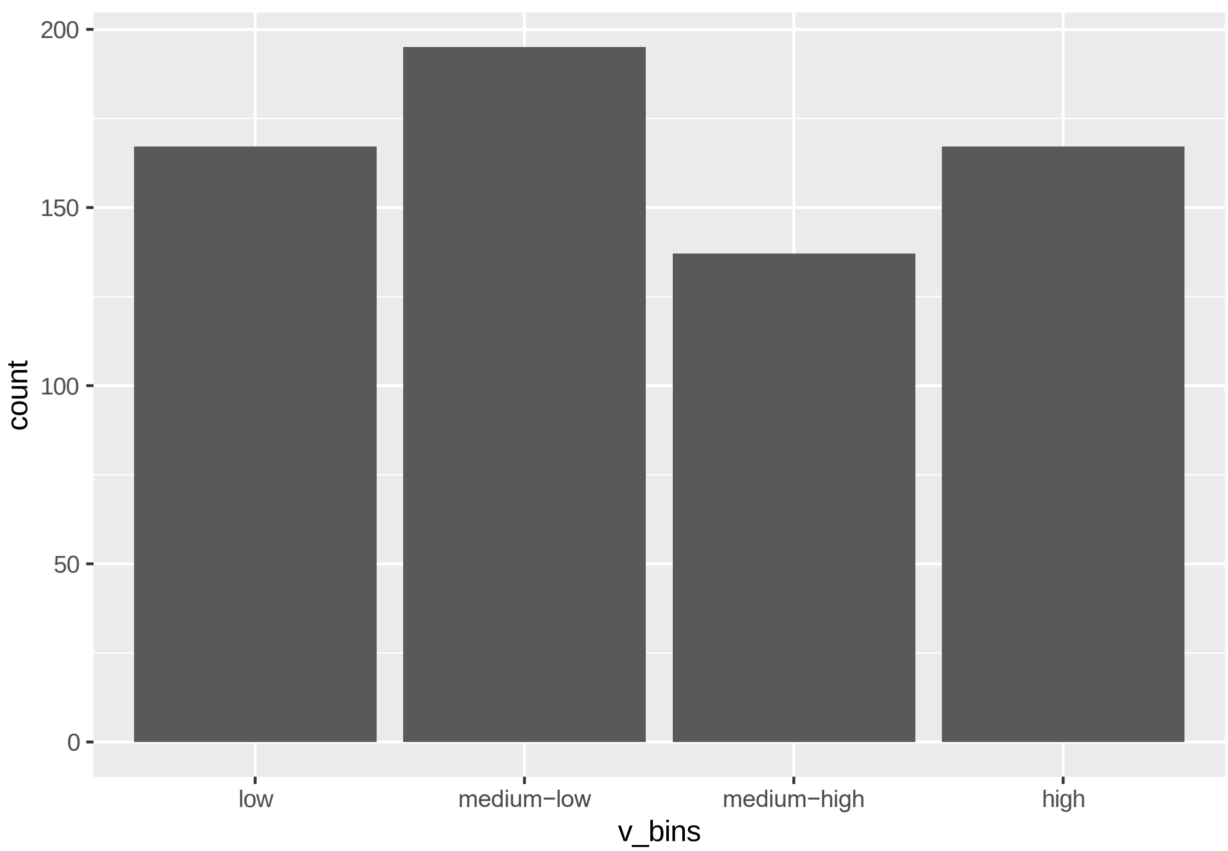
```
v<-bank_data$cont5
v<-as.data.frame(v)
```

Binning , as I learned in tutorial 2.

```
v <- v %>% mutate(v_bins = cut(v,breaks=bins,
                               labels=c("low","medium-low","medium-high","high")))
head(v)
```

```
##     v      v_bins
## 1 202 medium-high
## 2  43         low
## 3 280        high
## 4 100  medium-low
## 5 120  medium-low
## 6 360        high
```

```
ggplot(v, aes(x=v_bins)) + geom_bar()
```



we will smooth the above data with the mean of their respective bins.

```
low <- v %>%
  filter(v_bins=="low") %>%
  mutate(mean_value=mean(v,na.rm = T))

medium.low <- v %>%
  filter(v_bins=="medium-low") %>%
  mutate(mean_value=mean(v,na.rm = T))

medium.high <- v %>%
  filter(v_bins=="medium-high") %>%
  mutate(mean_value=mean(v,na.rm = T))

high <- v %>%
  filter(v_bins=="high") %>%
  mutate(mean_value=mean(v,na.rm = T))


v <- bind_rows(list(low, medium.low, medium.high,high))
```

```
head(v)
```

```
##    v v_bins mean_value
## 1 43    low   11.08383
## 2 52    low   11.08383
## 3  0    low   11.08383
## 4  0    low   11.08383
## 5  0    low   11.08383
## 6  0    low   11.08383
```

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.

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

starting with a fresh copy.

```
b <- bank_data
```

```
b$approval<- as.factor(b$approval)
```

```
class(b$approval)
```

```
## [1] "factor"
```

Evaluation method parameter. using cross validation with 10 folds

```r
train_control_cv = trainControl(method = "cv", number = 10)
```

Fit the model

```r
svm <- train(approval ~., data = b, method = "svmLinear",
             trControl  =  train_control_cv)
svm
```

```
## Support Vector Machines with Linear Kernel
##
## 666 samples
##  12 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 601, 599, 599, 600, 599, 599, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.863565  0.7290879
##
## Tuning parameter 'C' was held constant at a value of 1
```

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

Here we got the highest accuracy at C= 3.16 x 10^-3

```r
svm_grid <- train(approval ~., data = b, method = "svmLinear",
             trControl = train_control_cv, tuneGrid = grid)
svm_grid
```

```
## Support Vector Machines with Linear Kernel
##
## 666 samples
##  12 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 600, 599, 600, 599, 599, 600, ...
## 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  0.5705563  0.04777196
```

```
##     1.000000e-03   0.8378562   0.67004551
##     3.162278e-03   0.8676391   0.73724030
##     1.000000e-02   0.8646314   0.73137587
##     3.162278e-02   0.8631389   0.72834918
##     1.000000e-01   0.8631389   0.72834918
##     3.162278e-01   0.8631389   0.72834918
##     1.000000e+00   0.8631389   0.72834918
##     3.162278e+00   0.8631389   0.72834918
##     1.000000e+01   0.8631389   0.72834918
##     3.162278e+01   0.8631389   0.72834918
##     1.000000e+02   0.8631389   0.72834918
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.003162278.
```

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?
Yes, while training the model with SVM, the e1071 package in R sets the default parameter for C = 1 which controls the trade off between maximizing the margin and minimizing errors(increasing accuracy). Even if the default parameter of C=1, it is possible that the optimal value for C identified through tuning process could be different as it depends on the characteristics of data. From the above, we can see the optimal c value is 0.003167 for which there is a highest accuracy.

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

```
library(dplyr)
d  =  as.data.frame(starwars)
```

```
df=d
head(df)
```

```
##                  name height mass  hair_color    skin_color  eye_color  birth_year
## 1  Luke Skywalker     172    77         blond          fair       blue        19.0
## 2           C-3PO     167    75          <NA>          gold     yellow       112.0
## 3           R2-D2      96    32          <NA>   white, blue        red        33.0
## 4     Darth Vader     202   136          none         white     yellow        41.9
## 5     Leia Organa     150    49         brown         light      brown        19.0
## 6      Owen Lars      178   120   brown, grey         light       blue        52.0
##         sex    gender homeworld species ##
1    male masculine  Tatooine    Human ## 2
none  masculine    Tatooine     Droid   ## 3
none  masculine         Naboo     Droid ## 4
male  masculine    Tatooine      Human ## 5
female   feminine   Alderaan     Human ## 6
male  masculine   Tatooine     Human ##
```

```
## 1                                          The Empire Strikes Back, Revenge of the Sith, Return of
## 2                      The Empire Strikes Back, Attack of the Clones, The Phantom Menace, Revenge of t
## 3 The Empire Strikes Back, Attack of the Clones, The Phantom Menace, Revenge of the Sith, Return of
## 4                                                              The Empire Strikes Back, Revenge of t
## 5                                          The Empire Strikes Back, Revenge of the Sith, Return of
## 6                                                                                    Attack of the
##                           vehicles                starships
## 1 Snowspeeder, Imperial Speeder Bike X-wing, Imperial shuttle
## 2
## 3
## 4                                          TIE Advanced x1
## 5               Imperial Speeder Bike
## 6
```

```
drop <- c("films", "vehicles", "starships", "name")
df <- df[,!(names(starwars) %in% drop)]
```

```
summary(df)
```

```
##      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   :174.4   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
##
##
##
##
```

all NA's are dropperd.

```
df <- na.omit(df)
df[rowSums(is.na(df)) > 0, ]
```

```
##  [1] height     mass       hair_color skin_color eye_color  birth_year
##  [7] sex        gender     homeworld  species
## <0 rows> (or 0-length row.names)
```

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.

```
str(df)
```

```
## 'data.frame':    29 obs. of  10 variables:
##  $ height    : int  172 202 150 178 165 183 182 188 228 180 ...
##  $ mass      : num  77 136 49 120 75 84 77 84 112 80 ...
##  $ hair_color: chr  "blond" "none" "brown" "brown, grey" ...
##  $ skin_color: chr  "fair" "white" "light" "light" ...
##  $ eye_color : chr  "blue" "yellow" "brown" "blue" ...
##  $ birth_year: num 19 41.9 19 52 47 24 57 41.9 200 29 ...
##  $ sex       : chr  "male" "male" "female" "male" ...
##  $ gender    : chr  "masculine" "masculine" "feminine" "masculine" ...
##  $ homeworld : chr "Tatooine" "Tatooine" "Alderaan" "Tatooine" ...
##  $ species   : chr  "Human" "Human" "Human" "Human" ...
##  - attr(*, "na.action")= 'omit' Named int [1:58] 2 3 8 12 15 16 18 19 22 27 ...
##   ..- attr(*, "names")= chr [1:58] "2" "3" "8" "12" ...
```

```
dummy <- dummyVars(gender ~ ., data = df)
dummies <- as.data.frame(predict(dummy, newdata = df))
head(dummies)
```

```
##   height mass hair_colorauburn, white hair_colorblack hair_colorblond
## 1    172   77                       0               0               1
## 4    202  136                       0               0               0
## 5    150   49                       0               0               0
## 6    178  120                       0               0               0
## 7    165   75                       0               0               0
## 9    183   84                       0               1               0
##   hair_colorbrown hair_colorbrown, grey hair_colorgrey hair_colornone
## 1               0                     0              0              0
## 4               0                     0              0              1
## 5               1                     0              0              0
## 6               0                     1              0              0
## 7               1                     0              0              0
## 9               0                     0              0              0
##   hair_colorwhite skin_colorblue skin_colorbrown skin_colorbrown mottle
## 1               0              0               0                     0
## 4               0              0               0                     0
## 5               0              0               0                     0
## 6               0              0               0                     0
## 7               0              0               0                     0
## 9               0              0               0                     0
##   skin_colordark skin_colorfair skin_colorgreen skin_colorlight
## 1              0              1               0               0
## 4              0              0               0               0
## 5              0              0               0               1
## 6              0              0               0               1
## 7              0              0               0               1
## 9              0              0               0               1
##   skin_colororange skin_colorpale skin_colorred skin_colortan skin_colorunknown
```

```
## 1                  0                0                0                0                 0
## 4                  0                0                0                0                 0
## 5                  0                0                0                0                 0
## 6                  0                0                0                0                 0
## 7                  0                0                0                0                 0
## 9                  0                0                0                0                 0
##   skin_colorwhite skin_coloryellow eye_colorblack eye_colorblue
## 1               0                0              0             1
## 4               1                0              0             0
## 5               0                0              0             0
## 6               0                0              0             1
## 7               0                0              0             1
## 9               0                0              0             0
##   eye_colorblue-gray eye_colorbrown eye_colorhazel eye_colororange eye_colorred
## 1                  0              0              0               0            0
## 4                  0              0              0               0            0
## 5                  0              1              0               0            0
## 6                  0              0              0               0            0
## 7                  0              0              0               0            0
## 9                  0              1              0               0            0
##   eye_coloryellow birth_year sexfemale sexmale homeworldAlderaan
## 1               0       19.0         0       1                 0
## 4               1       41.9         0       1                 0
## 5               0       19.0         1       0                 1
## 6               0       52.0         0       1                 0
## 7               0       47.0         1       0                 0
## 9               0       24.0         0       1                 0
##   homeworldBespin homeworldCerea homeworldConcord Dawn homeworldCorellia
## 1               0              0                     0                 0
## 4               0              0                     0                 0
## 5               0              0                     0                 0
## 6               0              0                     0                 0
## 7               0              0                     0                 0
## 9               0              0                     0                 0
##   homeworldDathomir homeworldDorin homeworldEndor homeworldHaruun Kal
## 1                 0              0              0                   0
## 4                 0              0              0                   0
## 5                 0              0              0                   0
## 6                 0              0              0                   0
## 7                 0              0              0                   0
## 9                 0              0              0                   0
##   homeworldKamino homeworldKashyyyk homeworldMirial homeworldMon Cala
## 1               0                 0               0                0
## 4               0                 0               0                0
## 5               0                 0               0                0
## 6               0                 0               0                0
## 7               0                 0               0                0
## 9               0                 0               0                0
##   homeworldNaboo homeworldRyloth homeworldSerenno homeworldSocorro
## 1              0               0                0                0
## 4              0               0                0                0
## 5              0               0                0                0
## 6              0               0                0                0
## 7              0               0                0                0
```

```
## 9                    0                  0                   0                 0
##    homeworldStewjon homeworldTatooine homeworldTrandosha speciesCerean
## 1                   0                 1                  0             0
## 4                   0                 1                  0             0
## 5                   0                 0                  0             0
## 6                   0                 1                  0             0
## 7                   0                 1                  0             0
## 9                   0                 1                  0             0
##    speciesEwok speciesGungan speciesHuman speciesKel Dor speciesMirialan
## 1            0             0            1              0               0
## 4            0             0            1              0               0
## 5            0             0            1              0               0
## 6            0             0            1              0               0
## 7            0             0            1              0               0
## 9            0             0            1              0               0
##    speciesMon Calamari speciesTrandoshan speciesTwi'lek speciesWookiee
## 1                    0                 0              0              0
## 4                    0                 0              0              0
## 5                    0                 0              0              0
## 6                    0                 0              0              0
## 7                    0                 0              0              0
## 9                    0                 0              0              0
##    speciesZabrak
## 1              0
## 4              0
## 5              0
## 6              0
## 7              0
## 9              0
```

Adding target variable

```
dummy_for_svm = dummies
dummy_for_svm$gender = df$gender
colnames(dummy_for_svm)
```

```
##  [1] "height"                     "mass"
##  [3] "hair_colorauburn, white"    "hair_colorblack"
##  [5] "hair_colorblond"            "hair_colorbrown"
##  [7] "hair_colorbrown, grey"      "hair_colorgrey"
##  [9] "hair_colornone"             "hair_colorwhite"
## [11] "skin_colorblue"             "skin_colorbrown"
## [13] "skin_colorbrown mottle"     "skin_colordark"
## [15] "skin_colorfair"             "skin_colorgreen"
## [17] "skin_colorlight"            "skin_colororange"
## [19] "skin_colorpale"             "skin_colorred"
## [21] "skin_colortan"              "skin_colorunknown"
## [23] "skin_colorwhite"            "skin_coloryellow"
## [25] "eye_colorblack"             "eye_colorblue"
## [27] "eye_colorblue-gray"         "eye_colorbrown"
## [29] "eye_colorhazel"             "eye_colororange"
## [31] "eye_colorred"               "eye_coloryellow"
## [33] "birth_year"                 "sexfemale"
```

```
## [35] "sexmale"                  "homeworldAlderaan"
## [37] "homeworldBespin"          "homeworldCerea"
## [39] "homeworldConcord Dawn"    "homeworldCorellia"
## [41] "homeworldDathomir"        "homeworldDorin"
## [43] "homeworldEndor"           "homeworldHaruun Kal"
## [45] "homeworldKamino"          "homeworldKashyyyk"
## [47] "homeworldMirial"          "homeworldMon Cala"
## [49] "homeworldNaboo"           "homeworldRyloth"
## [51] "homeworldSerenno"         "homeworldSocorro"
## [53] "homeworldStewjon"         "homeworldTatooine"
## [55] "homeworldTrandosha"       "speciesCerean"
## [57] "speciesEwok"              "speciesGungan"
## [59] "speciesHuman"             "speciesKel Dor"
## [61] "speciesMirialan"          "speciesMon Calamari"
## [63] "speciesTrandoshan"        "speciesTwi'lek"
## [65] "speciesWookiee"           "speciesZabrak"
## [67] "gender"
```

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

```
preproc = c("center", "scale")
svm_gender <- train(gender ~ ., data = dummy_for_svm, method = "svmLinear", trControl = train_control_c
svm_gender
```

```
## Support Vector Machines with Linear Kernel
##
## 29 samples
## 66 predictors
##  2 classes: 'feminine', 'masculine'
##
## Pre-processing: centered (66), scaled (66)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 27, 25, 26, 27, 25, 26, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.975     0.9166667
##
## Tuning parameter 'C' was held constant at a value of 1
```

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.

removing near zero variance variable

```
dummy_for_pca <- dummies
nzv <- nearZeroVar(dummies)
length(nzv)
```

```
## [1] 39
```

```
dummy_for_pca <- dummy_for_pca[, -nzv]
head(dummy_for_pca)
```

```
##   height mass hair_colorblack hair_colorblond hair_colorbrown hair_colornone
## 1    172   77               0               1               0              0
## 4    202  136               0               0               0              1
## 5    150   49               0               0               1              0
## 6    178  120               0               0               0              0
## 7    165   75               0               0               1              0
## 9    183   84               1               0               0              0
##   hair_colorwhite skin_colordark skin_colorfair skin_colorlight
## 1               0              0              1               0
## 4               0              0              0               0
## 5               0              0              0               1
## 6               0              0              0               1
## 7               0              0              0               1
## 9               0              0              0               1
##   skin_colororange skin_colorpale skin_coloryellow eye_colorblue eye_colorbrown
## 1                0              0                0             1              0
## 4                0              0                0             0              0
## 5                0              0                0             0              1
## 6                0              0                0             1              0
## 7                0              0                0             1              0
## 9                0              0                0             0              1
##   eye_colorhazel eye_colororange eye_coloryellow birth_year sexfemale sexmale
## 1              0               0               0       19.0         0       1
## 4              0               0               1       41.9         0       1
## 5              0               0               0       19.0         1       0
## 6              0               0               0       52.0         0       1
## 7              0               0               0       47.0         1       0
## 9              0               0               0       24.0         0       1
##   homeworldCorellia homeworldMirial homeworldNaboo homeworldTatooine
## 1                 0               0              0                 1
## 4                 0               0              0                 1
## 5                 0               0              0                 0
## 6                 0               0              0                 1
## 7                 0               0              0                 1
## 9                 0               0              0                 1
##     speciesHuman speciesMirialan
## 1              1               0
## 4              1               0
## 5              1               0
## 6              1               0
## 7              1               0
## 9              1               0
```

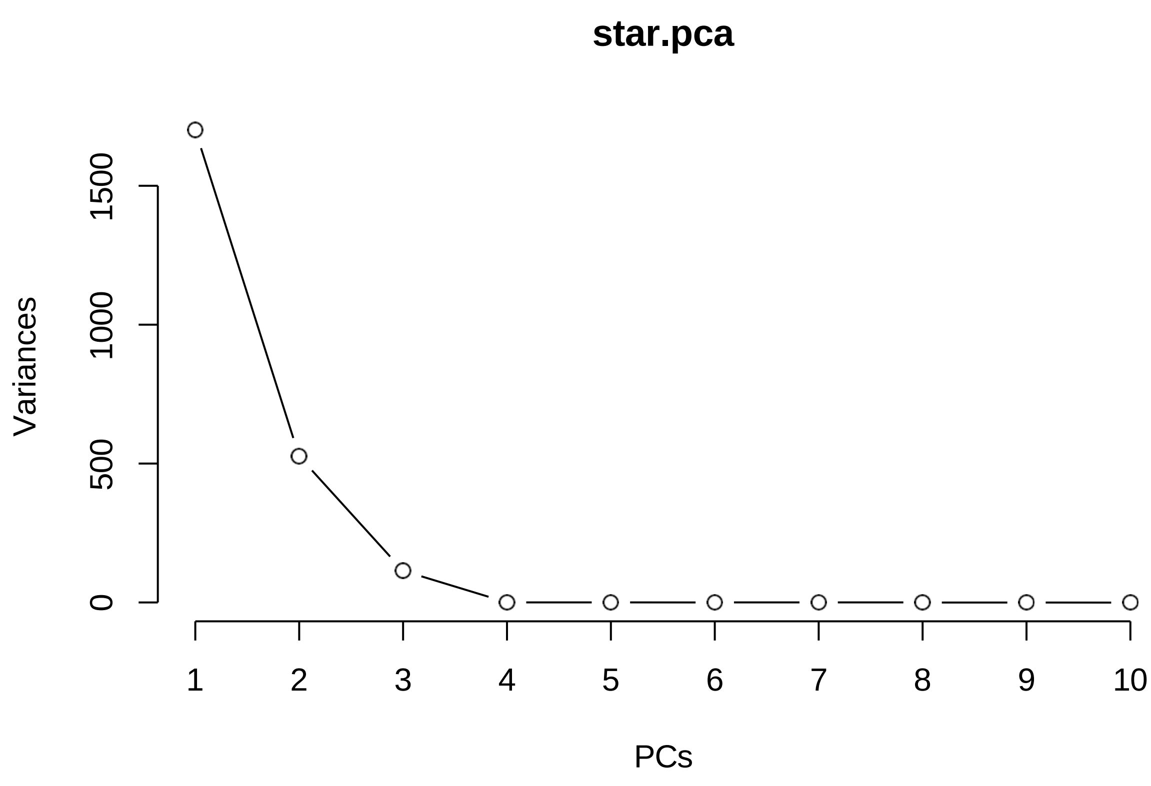
```
#pca object
star.pca <- prcomp(dummy_for_pca)
summary(star.pca)
```

```
## Importance of components:
##                              PC1     PC2      PC3     PC4     PC5     PC6     PC7
```

```
## Standard deviation       41.2487  22.9459  10.69652  0.73879  0.72440  0.56879  0.55673
## Proportion of Variance    0.7256   0.2245   0.04879  0.00023  0.00022  0.00014  0.00013
## Cumulative Proportion     0.7256   0.9501   0.99886  0.99909  0.99932  0.99946  0.99959
##                              PC8      PC9      PC10     PC11     PC12     PC13     PC14
## Standard deviation       0.46260  0.42802  0.33819  0.30140  0.26756  0.25373  0.24398
## 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  0.99988  0.99990  0.99993
##                             PC15     PC16     PC17     PC18     PC19    PC20     PC21
## Standard deviation       0.20403  0.18732  0.17363  0.14595  0.11313  0.1039  0.08506
## Proportion of Variance   0.00002  0.00001  0.00001  0.00001  0.00001  0.0000  0.00000
## Cumulative Proportion    0.99995  0.99996  0.99997  0.99998  0.99999  1.0000  1.00000
##                             PC22    PC23     PC24      PC25      PC26      PC27
## Standard deviation       0.07144  0.0581  0.03561  2.748e-15  2.748e-15  2.748e-15
## Proportion of Variance   0.00000  0.0000  0.00000  0.000e+00  0.000e+00  0.000e+00
## Cumulative Proportion    1.00000  1.0000  1.00000  1.000e+00  1.000e+00  1.000e+00
```

from the graph, we can observe that most of the variance is captured by 3 PC's. We can model our data using 3 PCA

```
screeplot(star.pca, type = "l") + title(xlab = "PCs")
```




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

```r
preProc_pca <- preProcess(dummy_for_pca, method="pca", pcaComp=3)
star.pc  <-  predict(preProc_pca,dummy_for_pca)
```

```r
star.pc$gender<-df$gender
head(star.pc)
```

```
##          PC1        PC2        PC3    gender
## 1  0.6974461 -1.6519796 -3.3727735 masculine
## 4  2.2907517  1.3527164 -1.6986150 masculine
## 5 -1.9439564 -2.8142749  2.1652893  feminine
## 6  0.6910936 -0.7117651 -2.3621359 masculine
## 7 -1.7888898 -2.0699620 -0.5901683  feminine
## 9  0.1674467 -1.6602523 -0.8266389 masculine
```

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.

Train-Test split

```r
set.seed(123)
index = createDataPartition(y=star.pc$gender, p=0.7, list=FALSE)
train_set = star.pc[index,]
test_set = star.pc[-index,]
```

Building SVM model with the train test partitioning method

```r
#fit the model
svm_train_test_split <- train(gender ~., data = train_set, method = "svmLinear",tuneGrid=grid)
#predict on test set
pred_split <- predict(svm_train_test_split, test_set)
```

here we can see the tuning parameter was set to 10 by grid search.

```r
svm_train_test_split
```

```
## 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  0.7735368  0.0000000
##   3.162278e-05  0.7735368  0.0000000
##   1.000000e-04  0.7735368  0.0000000
```

```
##     3.162278e-04  0.7735368  0.0000000
##     1.000000e-03  0.7735368  0.0000000
##     3.162278e-03  0.7735368  0.0000000
##     1.000000e-02  0.7735368  0.0000000
##     3.162278e-02  0.8528874  0.3174095
##     1.000000e-01  0.9008557  0.5345817
##     3.162278e-01  0.9367937  0.7401767
##     1.000000e+00  0.9559524  0.8442205
##     3.162278e+00  0.9942857  0.9545455
##     1.000000e+01  0.9942857  0.9545455
##     3.162278e+01  0.9942857  0.9545455
##     1.000000e+02  0.9942857  0.9545455
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 3.162278.
```

```r
#accuracy
sum(pred_split == test_set$gender) / nrow(test_set)
```

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

Using Bootstrap

```r
train_control_boot = trainControl(method = "boot", number = 100)
svm_bootstrap <- train(gender ~., data = star.pc, method = "svmLinear",
                trControl = train_control_boot,tuneGrid = grid)
svm_bootstrap
```

```
## Support Vector Machines with Linear Kernel
##
## 29 samples
##  3 predictor
##  2 classes: 'feminine', 'masculine'
##
## No pre-processing
## Resampling: Bootstrapped (100 reps)
## Summary of sample sizes: 29, 29, 29, 29, 29, 29, ...
## Resampling results across tuning parameters:
##
##    C             Accuracy   Kappa
##    1.000000e-05  0.8143122  0.0000000
##    3.162278e-05  0.8143122  0.0000000
##    1.000000e-04  0.8143122  0.0000000
##    3.162278e-04  0.8143122  0.0000000
##    1.000000e-03  0.8143122  0.0000000
##    3.162278e-03  0.8143122  0.0000000
##    1.000000e-02  0.8143122  0.0000000
##    3.162278e-02  0.8577268  0.3088425
##    1.000000e-01  0.8727622  0.4170452
##    3.162278e-01  0.8982470  0.5677483
##    1.000000e+00  0.9200683  0.6944700
##    3.162278e+00  0.9249024  0.7220528
##    1.000000e+01  0.9235826  0.7247733
```

```
##     3.162278e+01    0.9235826    0.7247733
##     1.000000e+02    0.9235826    0.7247733
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 3.162278.
```

confusion matrix

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

```
## 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
##
```

e. Whether or not it has improved the accuracy, what has PCA done for the complexity of the model? Here, after apply svm on PCA data, it accuracy got decreased due to reduction in the information of the variables. However, in Tutorial 2, I learned that sometimes the accuracy gets reduced but its is a good practice to have a PCA as it generalize the model. PCA has reduced the complexity of the model.

Problem 4 (Bonus)

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

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?

```
data("Sacramento")
```

```
df_sac = as.data.frame(Sacramento)
head(df_sac)
```

```
##          city    zip beds baths sqft        type price latitude longitude
## 1 SACRAMENTO z95838    2     1  836 Residential 59222 38.63191 -121.4349
## 2 SACRAMENTO z95823    3     1 1167 Residential 68212 38.47890 -121.4310
## 3 SACRAMENTO z95815    2     1  796 Residential 68880 38.61830 -121.4438
## 4 SACRAMENTO z95815    2     1  852 Residential 69307 38.61684 -121.4391
## 5 SACRAMENTO z95824    2     1  797 Residential 81900 38.51947 -121.4358
## 6 SACRAMENTO z95841    3     1 1122       Condo 89921 38.66260 -121.3278
```

```
df_sac <-select(df_sac, -c(city,zip))
head(df_sac)
```

```
##   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
```

```
str(df_sac)
```

```
## 'data.frame':    932 obs. of  7 variables:
##  $ beds     : int  2 3 2 2 2 3 3 3 2 3 ...
##  $ baths    : num  1 1 1 1 1 1 2 1 2 2 ...
##  $ sqft     : int  836 1167 796 852 797 1122 1104 1177 941 1146 ...
##  $ type     : Factor w/ 3 levels "Condo","Multi_Family",..: 3 3 3 3 3 1 3 3 1 3 ...
##  $ price    : int  59222 68212 68880 69307 81900 89921 90895 91002 94905 98937 ...
##  $ latitude : num  38.6 38.5 38.6 38.6 38.5 ...
##  $ longitude: num  -121 -121 -121 -121 -121 ...
```

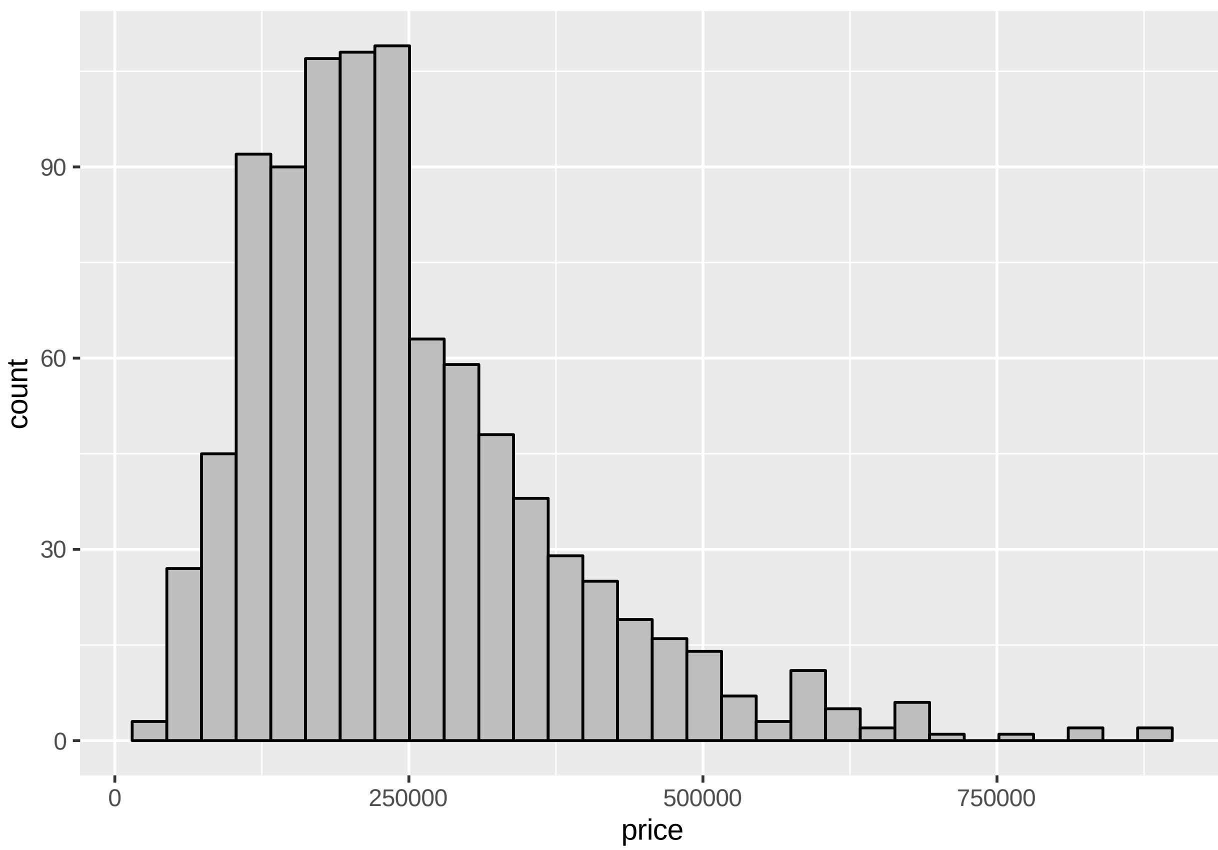
From the summary we can see, type variable has 3 category. Beds, baths, sqft has nearly normal distribution and price is right skewed.

```
summary(df_sac)
```

```
##       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
```

```
ggplot(df_sac,aes(x=price)) + geom_histogram(color= "black" , fill="grey")
```

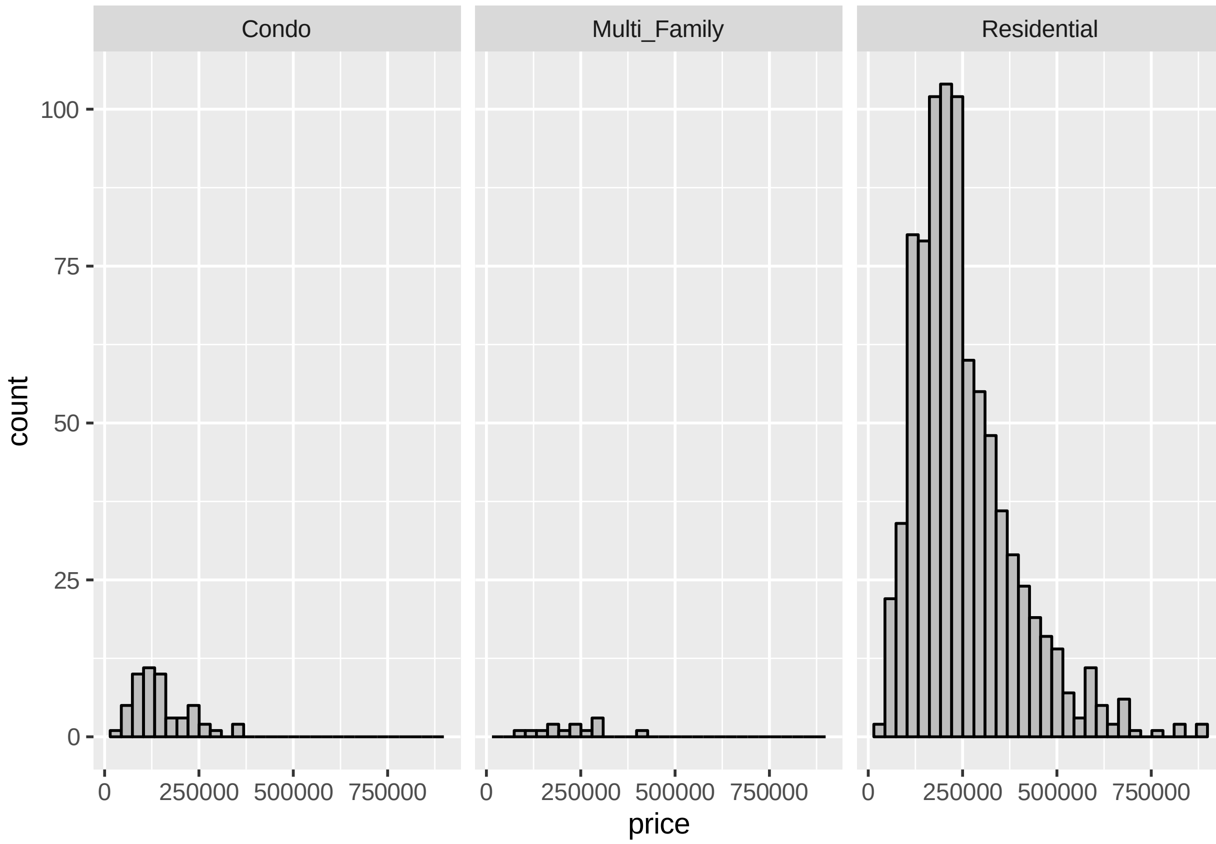


Yes , we hace a class imbalance as there are relatively few instances if class Condo and Multi_Family compared to Residential.

```
table(df_sac$type)
```

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

```
ggplot(df_sac,aes(x=price)) + geom_histogram(color= "black" , fill="grey") +
  facet_wrap(~ type)
```



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.

Here I am trying with (1)removing other variables that you know should not be part of the prediction. the columns latitude and longitude are not needed.

```
df_sac <-select(df_sac, -c(latitude,longitude))
head(df_sac)
```

```
##   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.

```
set.seed(123)
index_sac= createDataPartition(y=df_sac$type, p=0.7, list=FALSE)
train_set_sac = df_sac[index_sac,]
test_set_sac = df_sac[-index_sac,]
```

```
svm_split_sac <- train(type ~., data = train_set_sac, method = "svmLinear", preProcess=preproc
                       ,tuneGrid=grid)
pred_split_sac <- predict(svm_split_sac, test_set_sac)
```

```
svm_split_sac
```

```
## Support Vector Machines with Linear Kernel
##
## 655 samples
##   4 predictor
##   3 classes: 'Condo', 'Multi_Family', 'Residential'
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 655, 655, 655, 655, 655, 655, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.9298792  0.00000000
##   3.162278e-05  0.9298792  0.00000000
##   1.000000e-04  0.9298792  0.00000000
##   3.162278e-04  0.9298792  0.00000000
##   1.000000e-03  0.9298792  0.00000000
##   3.162278e-03  0.9298792  0.00000000
##   1.000000e-02  0.9298792  0.00000000
##   3.162278e-02  0.9298792  0.00000000
##   1.000000e-01  0.9298792  0.00000000
##   3.162278e-01  0.9302258  0.02157768
##   1.000000e+00  0.9307211  0.03990870
##   3.162278e+00  0.9313789  0.05530125
##   1.000000e+01  0.9313789  0.05530125
##   3.162278e+01  0.9313789  0.05530125
##   1.000000e+02  0.9315428  0.05866388
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 100.
```

```
sum(pred_split_sac == test_set_sac$type) / nrow(test_set_sac)
```

```
## [1] 0.9350181
```

Here, we can see the accuracy is high but the value of kappa is 0 which shows the classifier has not performed well on classifying the type. And there was class imbalance also.

```
confusionMatrix(test_set_sac$type, pred_split_sac)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Condo Multi_Family Residential
##   Condo             0            0          15
##   Multi_Family      0            0           3
##   Residential       0            0         259
##
## Overall Statistics
##
##                Accuracy : 0.935
##                  95% CI : (0.8992, 0.961)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Condo Class: Multi_Family Class: Residential
## Sensitivity                    NA                  NA              0.935
## Specificity               0.94585             0.98917                 NA
## Pos Pred Value                 NA                  NA                 NA
## Neg Pred Value                 NA                  NA                 NA
## Prevalence                0.00000             0.00000              1.000
## Detection Rate            0.00000             0.00000              0.935
## Detection Prevalence      0.05415             0.01083              0.935
## Balanced Accuracy              NA                  NA                 NA
```

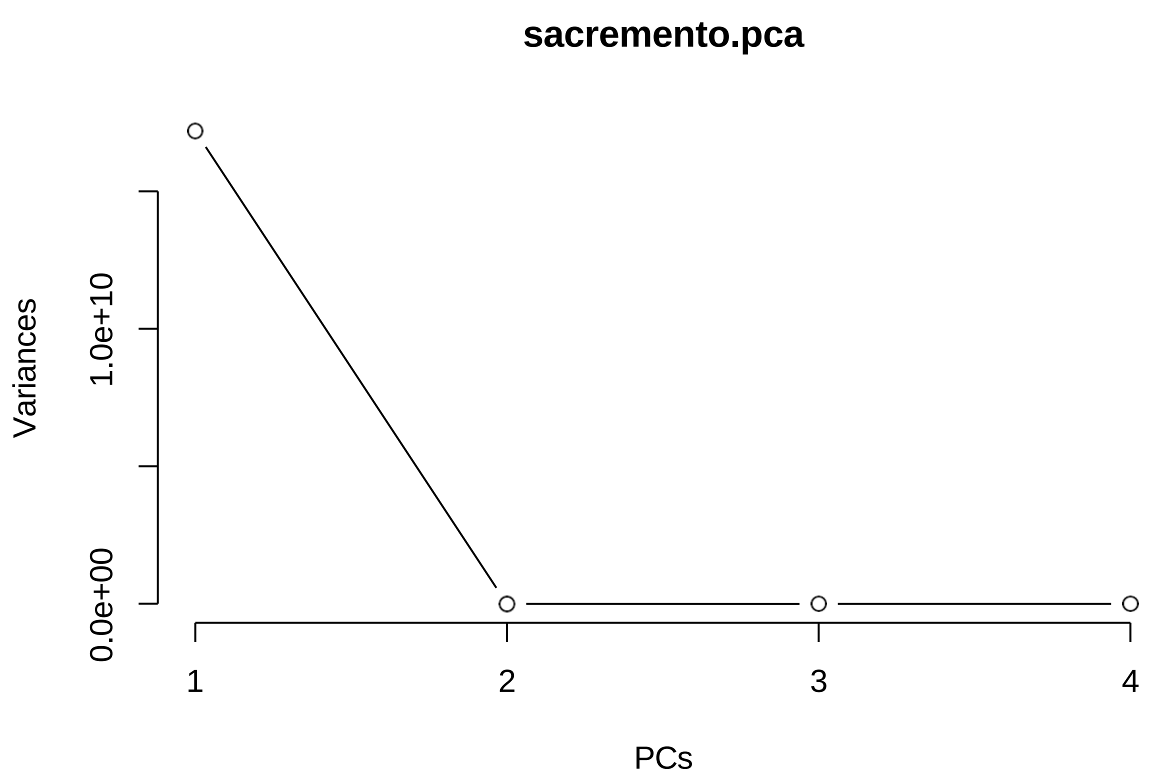
d. Return to (b) and try at least one other way to try to improve the data before running SVM again, as in (c).

```
df_pca = select(df_sac, -c(type))
```

```
sacremento.pca <- prcomp(df_pca)
summary(sacremento.pca)
```

```
## 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
```

```
screeplot(sacremento.pca, type = "l") + title(xlab = "PCs")
```




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

```
preProc_sac <- preProcess(df_pca, method="pca", pcaComp=2)
sacremento.pc <- predict(preProc_sac, df_pca)
sacremento.pc$type<-df_sac$type
```

```
svm_split_sac_pc <- train(type ~., data = sacremento.pc, method = "svmLinear",trControl = train_control
svm_split_sac_pc
```

```
## Support Vector Machines with Linear Kernel
##
## 932 samples
##   2 predictor
##   3 classes: 'Condo', 'Multi_Family', 'Residential'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 838, 839, 839, 840, 840, 838, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.9292123  0
```

```
##      3.162278e-05   0.9292123   0
##      1.000000e-04   0.9292123   0
##      3.162278e-04   0.9292123   0
##      1.000000e-03   0.9292123   0
##      3.162278e-03   0.9292123   0
##      1.000000e-02   0.9292123   0
##      3.162278e-02   0.9292123   0
##      1.000000e-01   0.9292123   0
##      3.162278e-01   0.9292123   0
##      1.000000e+00   0.9292123   0
##      3.162278e+00   0.9292123   0
##      1.000000e+01   0.9292123   0
##      3.162278e+01   0.9292123   0
##      1.000000e+02   0.9292123   0
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1e-05.
```

We can observe the accuracy and kappa which is same and the classifier is not classifying the minority class.

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.

data from smaller classes

```
small = df_sac[df_sac$type=="Condo" | df_sac$type=="Multi_Family",]
```

data from smaller classes

```
large= df_sac[df_sac$type=="Residential", ]
```

sample data from large class

```
large_sample = large[sample(nrow(large),40),]
```

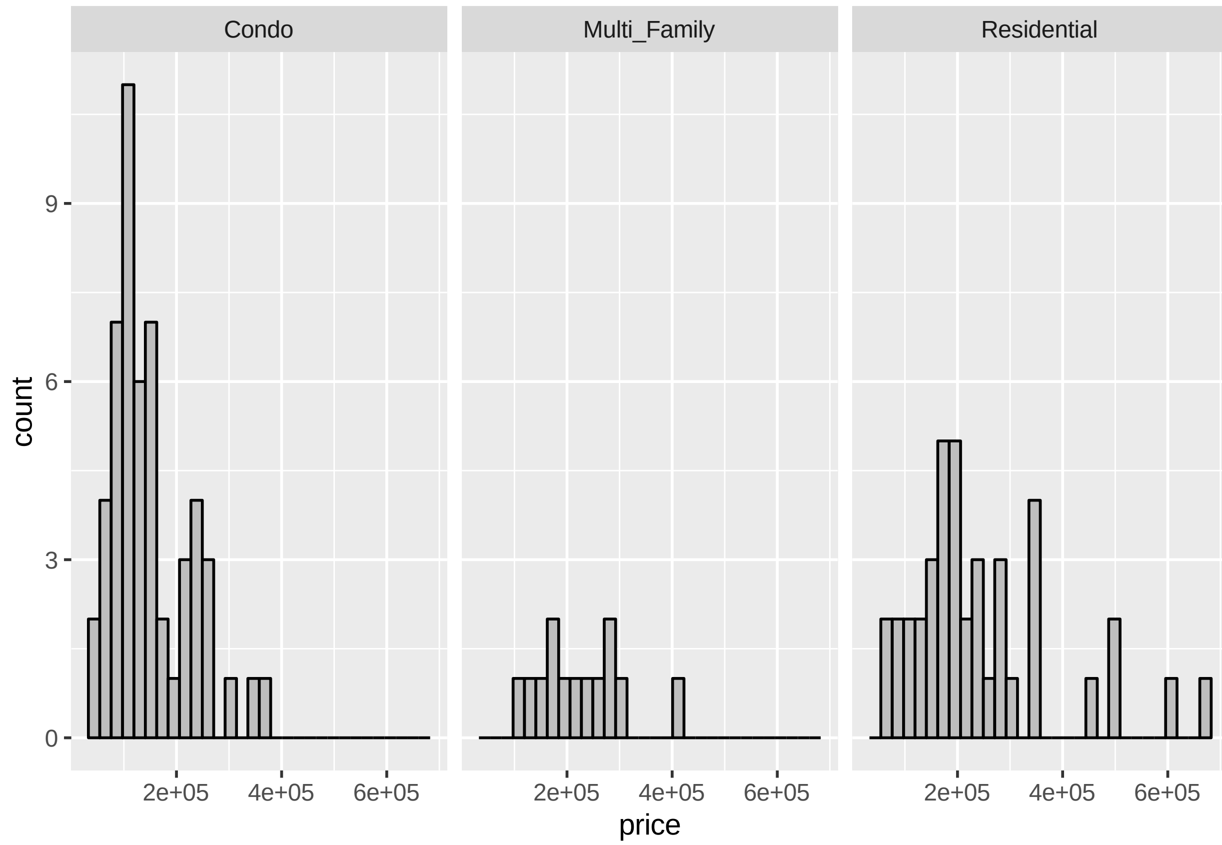
combining data

```
new_data = rbind(small,large_sample)
summary(new_data)
```

```
##        beds           baths            sqft                  type
##  Min.   :1.000   Min.   :1.000   Min.   : 484.0   Condo         :53
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.: 944.8   Multi_Family:13
##  Median :3.000   Median :2.000   Median :1151.5   Residential  :40
##  Mean   :2.755   Mean   :1.906   Mean   :1441.0
##  3rd Qu.:4.000   3rd Qu.:2.000   3rd Qu.:1790.2
##  Max.   :8.000   Max.   :4.000   Max.   :3705.0
##       price
##  Min.   : 40000
```

1st Qu.:115000
Median :169000
Mean :193327
3rd Qu.:239925 ##
Max. :668365

```
ggplot(new_data,aes(x=price)) + geom_histogram(color= "black" , fill="grey") +
  facet_wrap(~ type)
```




```
set.seed(123)
index_new_data= createDataPartition(y=new_data$type, p=0.7, list=FALSE)
train_set_new = new_data[index_new_data,]
test_set_new = new_data[-index_new_data,]
```

```
svm_split_new_data <- train(type ~., data = train_set_new, method = "svmLinear", preProcess=preproc, tu
pred_split_new <- predict(svm_split_new_data, test_set_new)
```

```
svm_split_new_data
```

```
## Support Vector Machines with Linear Kernel
##
## 76 samples
##  4 predictor
##  3 classes: 'Condo', 'Multi_Family', 'Residential'
```

```
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 76, 76, 76, 76, 76, 76, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-05  0.4943461  0.00000000
##   3.162278e-05  0.4943461  0.00000000
##   1.000000e-04  0.4943461  0.00000000
##   3.162278e-04  0.4943461  0.00000000
##   1.000000e-03  0.4943461  0.00000000
##   3.162278e-03  0.5094580  0.02497256
##   1.000000e-02  0.5892143  0.20888116
##   3.162278e-02  0.6588605  0.37684206
##   1.000000e-01  0.6955527  0.45790418
##   3.162278e-01  0.7500133  0.56197190
##   1.000000e+00  0.7550134  0.57517416
##   3.162278e+00  0.7453505  0.56138009
##   1.000000e+01  0.7413466  0.55461788
##   3.162278e+01  0.7386519  0.55027449
##   1.000000e+02  0.7316451  0.53900728
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.
```

```
sum(pred_split_new == test_set_new$type) / nrow(test_set_new)
```

```
## [1] 0.9
```

```
confusionMatrix(test_set_new$type, pred_split_new)
```

```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction     Condo Multi_Family Residential
##   Condo           14            0           1
##   Multi_Family     1            2           0
##   Residential      1            0          11
##
## Overall Statistics
##
##                Accuracy : 0.9
##                  95% CI : (0.7347, 0.9789)
##     No Information Rate : 0.5333
##     P-Value [Acc > NIR] : 1.989e-05
##
##                   Kappa : 0.8235
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: Condo Class: Multi_Family Class: Residential
## Sensitivity                0.8750             1.00000              0.9167
## Specificity                0.9286             0.96429              0.9444
## Pos Pred Value             0.9333             0.66667              0.9167
## Neg Pred Value             0.8667             1.00000              0.9444
## Prevalence                 0.5333             0.06667              0.4000
## Detection Rate             0.4667             0.06667              0.3667
## Detection Prevalence       0.5000             0.10000              0.4000
## Balanced Accuracy          0.9018             0.98214              0.9306
```

after sampling the entire data set and making type variable near to balance, we can observe its accuracy and confusion matrix report, which tells us how the accuracy got increased.

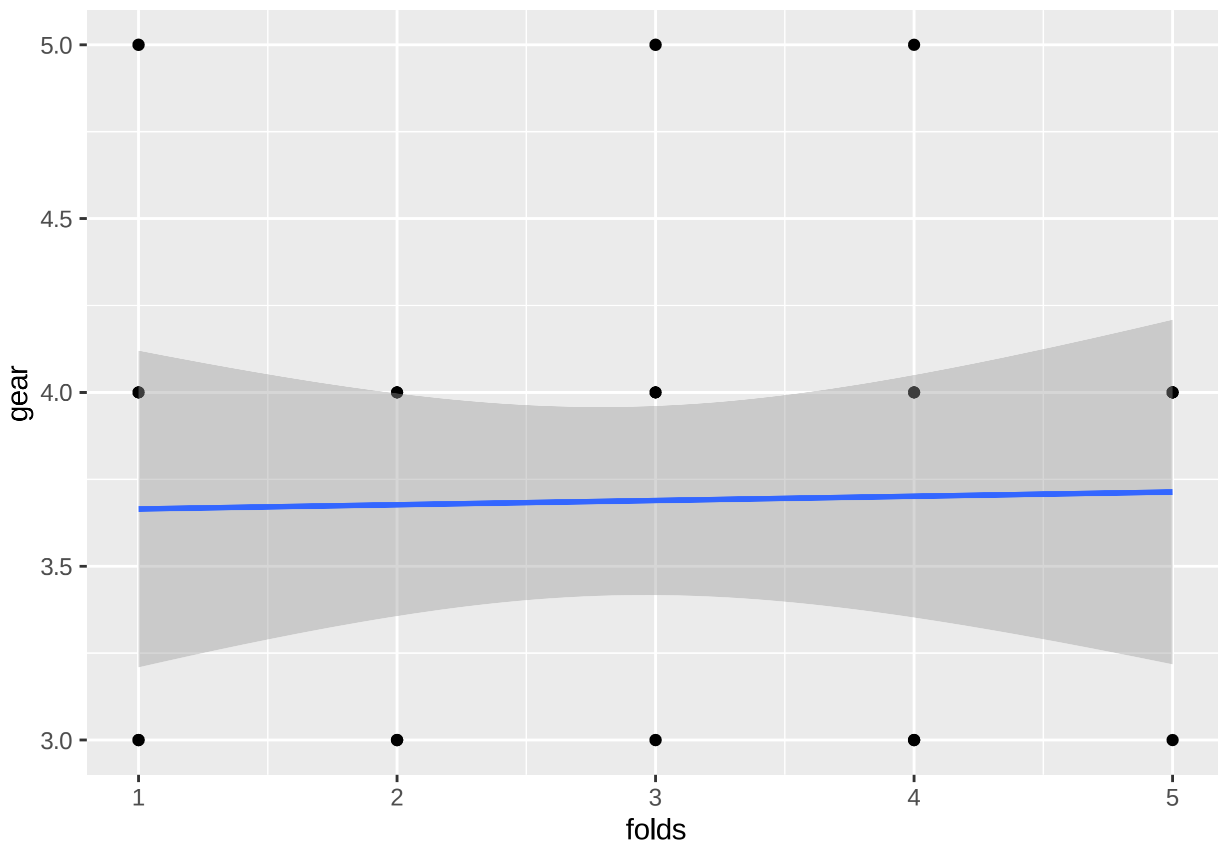
Problem 5 (Bonus)

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
mycars$folds = 0
flds = createFolds(1:nrow(mycars), k=5, list=TRUE)
```

```
 for (i in 1:5)
 {
   mycars$folds[flds[[i]]] = i
  }
```

```
ggplot(mycars, aes(folds, gear)) + geom_point() + geom_smooth(method = lm)
```


5.0
4.5
4.0
3.5
3.0
gear
1
2
3
4
5
folds