DSC 44- Homework 2

Sanchal Dhurve

1.For this problem, you will load and perform some cleaning steps on a dataset in the provided Bank-Data.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.

```r
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.4.2

## Warning: package 'ggplot2' was built under R version 4.4.2

## Warning: package 'tibble' was built under R version 4.4.2

## Warning: package 'tidyr' was built under R version 4.4.2

## Warning: package 'readr' was built under R version 4.4.2

## Warning: package 'purrr' was built under R version 4.4.2

## Warning: package 'dplyr' was built under R version 4.4.2

## Warning: package 'stringr' was built under R version 4.4.2

## Warning: package 'forcats' was built under R version 4.4.2

## Warning: package 'lubridate' was built under R version 4.4.2

## -- 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 1.9.4     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 error
```

```r
library(ggplot2)
library(dplyr)
library(readr)
library(tidyr)
library(ggthemes)
```

```
## Warning: package 'ggthemes' was built under R version 4.4.2
```

```r
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 4.4.2
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```r
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.4.2
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
#load the bankData.csv file
bankdata <- read.csv("C:/Users/SDHURVE/Documents/bankdata.csv", header = TRUE)

head(bankdata)
```

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

```r
str(bankdata)
```

```
## '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 ...
```

```r
summary(bankdata)
```

```
##        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
##
```

```r
# Check for missing values (Numerical Count)
colSums(is.na(bankdata))
```

```
##            X        cont1        cont2        cont3        bool1        bool2
##            0           12            0            0            0            0
##        cont4        bool3        cont5        cont6     approval credit.score
##            0            0           13            0            0            0
##         ages
##            0
```

```r
# Indicating the indices of missing values
which(is.na(bankdata), arr.ind = TRUE)
```

```
##       row col
##  [1,]  84   2
##  [2,]  87   2
##  [3,]  93   2
##  [4,]  98   2
##  [5,] 255   2
##  [6,] 287   2
##  [7,] 330   2
##  [8,] 446   2
##  [9,] 451   2
## [10,] 501   2
## [11,] 516   2
## [12,] 609   2
## [13,]  72   9
## [14,] 203   9
## [15,] 207   9
## [16,] 244   9
## [17,] 271   9
## [18,] 279   9
## [19,] 331   9
## [20,] 407   9
## [21,] 446   9
## [22,] 457   9
## [23,] 593   9
## [24,] 623   9
## [25,] 627   9
```

```r
# Remove rows with missing values
bankdata <- na.omit(bankdata)

# Verify missing values are removed
sum(is.na(bankdata))
```

```
## [1] 0
```

```r
# Check structure of cleaned dataset
str(bankdata)
```

```
## '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       : 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 ...
##  - 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" ...
```

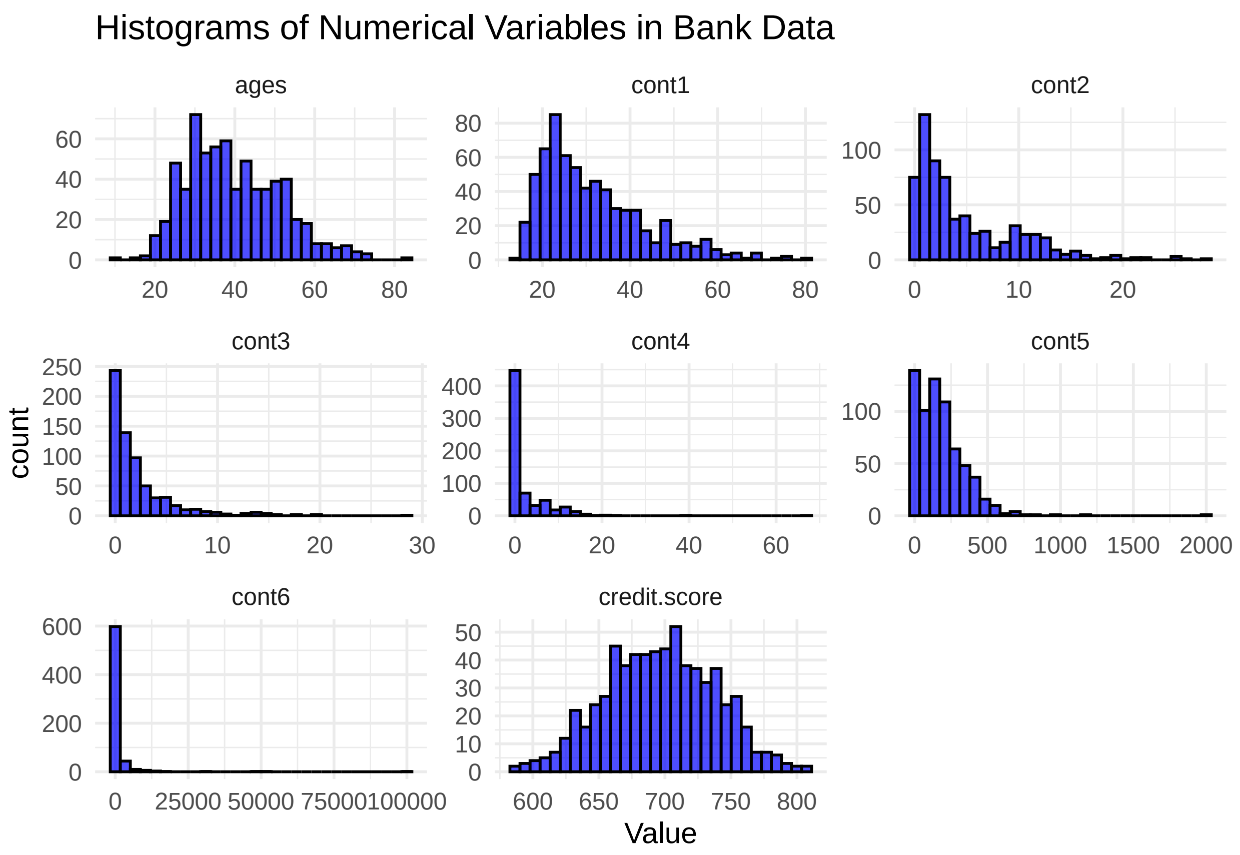
```
# Summary statistics of the cleaned dataset
summary(bankdata)
```

```
##        X             cont1           cont2            cont3
##  Min.   :  1.0   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:172.2   1st Qu.:22.60   1st Qu.: 1.010   1st Qu.: 0.165
##  Median :347.5   Median :28.50   Median : 2.750   Median : 1.000
##  Mean   :345.8   Mean   :31.57   Mean   : 4.798   Mean   : 2.222
##  3rd Qu.:519.8   3rd Qu.:38.25   3rd Qu.: 7.207   3rd Qu.: 2.585
##  Max.   :690.0   Max.   :80.25   Max.   :28.000   Max.   :28.500
##     bool1              bool2               cont4           bool3
##  Length:666         Length:666         Min.   : 0.000   Length:666
##  Class :character   Class :character   1st Qu.: 0.000   Class :character
##  Mode  :character   Mode  :character   Median : 0.000   Mode  :character
##                                        Mean   : 2.459
##                                        3rd Qu.: 3.000
##                                        Max.   :67.000
##      cont5              cont6              approval          credit.score
##  Min.   :   0.00   Min.   :     0.0   Length:666         Min.   :585.1
##  1st Qu.:  75.25   1st Qu.:     0.0   Class :character   1st Qu.:666.4
##  Median : 160.00   Median :     5.0   Mode  :character   Median :697.1
##  Mean   : 182.12   Mean   :   998.6                      Mean   :696.3
##  3rd Qu.: 271.00   3rd Qu.:   399.0                      3rd Qu.:726.4
##  Max.   :2000.00   Max.   :100000.0                      Max.   :806.0
##       ages
##  Min.   :11.0
##  1st Qu.:31.0
##  Median :38.0
##  Mean   :39.7
##  3rd Qu.:48.0
##  Max.   :84.0
```

```
# Convert boolean variables to factors

bankdata <- bankdata %>%
  mutate(
    bool1 = as.factor(bool1),
    bool2 = as.factor(bool2),
    bool3 = as.factor(bool3),
    approval = as.factor(approval))
```

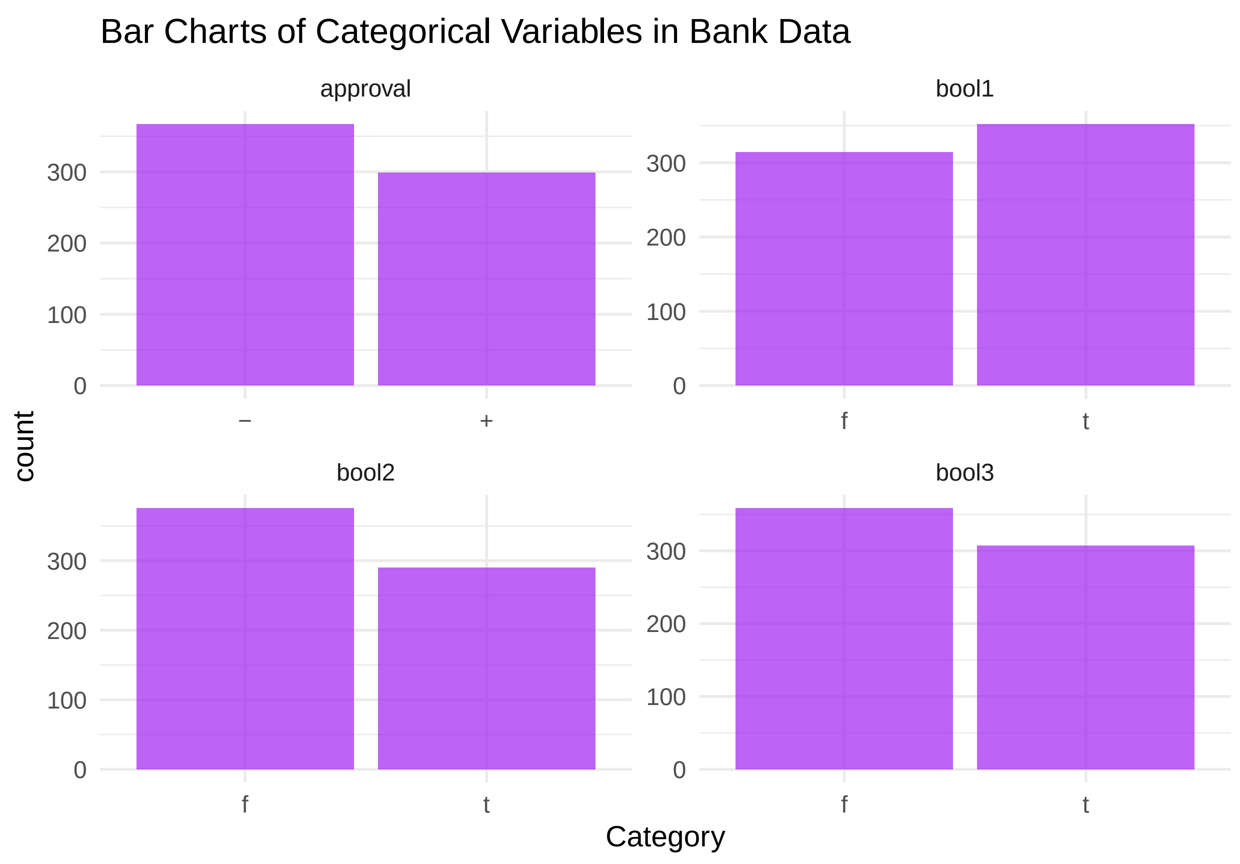
```
# Remove index column before plotting
bankdata <- bankdata %>% select(-X)

# Identify numeric variables
numeric_vars <- sapply(bankdata, is.numeric)
numeric_data <- bankdata[, numeric_vars]
```

```
# Select only numeric variables for histograms
numeric_data <- bankdata %>% select(where(is.numeric))

# Convert data to long format for facet_wrap()
numeric_long <- numeric_data %>% pivot_longer(cols = everything(), names_to = "Variable", values_to = "V

# Plot histograms using facet_wrap()
ggplot(numeric_long, aes(x = Value)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.7, color = "black") +
  facet_wrap(~Variable, scales = "free") +  # Wrap multiple histograms in one figure
  labs(title = "Histograms of Numerical Variables in Bank Data") +
  theme_minimal()
```



```
# Select only categorical variables for bar charts
categorical_data <- bankdata %>% select(where(is.factor))

# Convert data to long format for facet_wrap()
categorical_long <- categorical_data %>% pivot_longer(cols = everything(), names_to = "Variable", values
```

```r
# Plot bar charts using facet_wrap()
ggplot(categorical_long, aes(x = Category)) +
  geom_bar(fill = "purple", alpha = 0.7) +
  facet_wrap(~Variable, scales = "free") +
  labs(title = "Bar Charts of Categorical Variables in Bank Data") +
  theme_minimal()
```

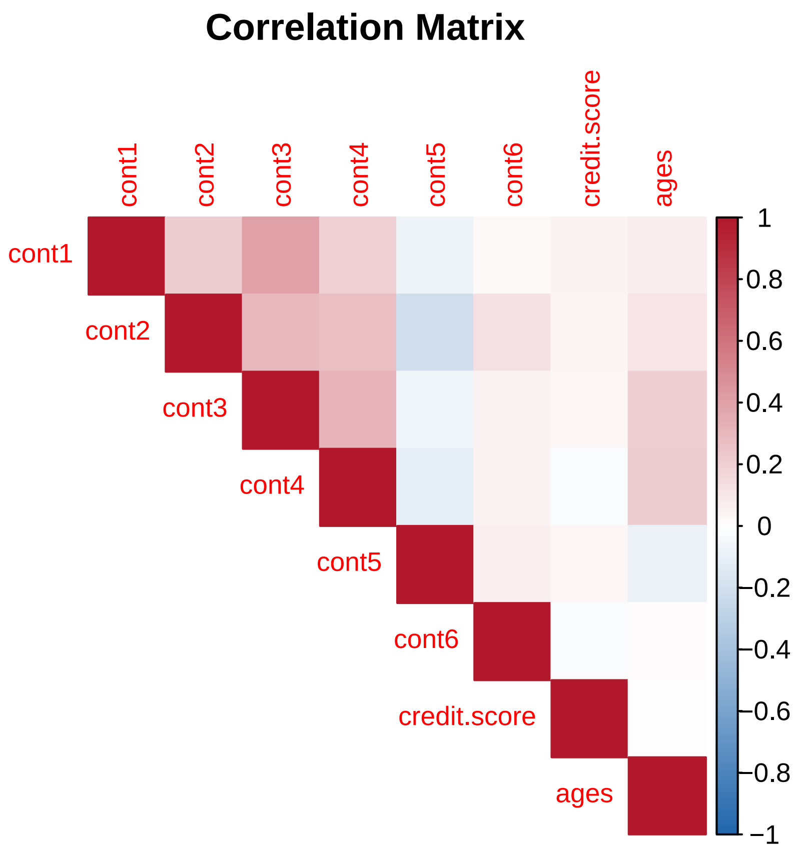


```r
#Load necessary library
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.4.2
```

```
## corrplot 0.95 loaded
```

```r
#Compute correlation matrix
cor_matrix <- cor(numeric_data, use = "complete.obs")

#Define a better color palette with improved contrast
col_scheme <- colorRampPalette(c("#2166AC", "white", "#B2182B"))(200)

#Plot using corrplot with enhanced colors
corrplot(cor_matrix, method = "color", type = "upper",
 col = col_scheme, tl.cex = 0.8, title = "Correlation Matrix", mar = c(0, 0, 2, 0))
```



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

Based on the tutorials- Z-score normalization is used to center the data around 0 and scale it to have unit variance. This is useful for data with normally distributed values.

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.2

## Loading required package: lattice

##
## Attaching package: 'caret'

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

```r
#Selecting cont4 for Z-score normalization
zScore <- subset(bankdata, select = c(cont4))

#Preprocess using center and scale
preProcessCont4 <- preProcess(as.data.frame(zScore), method = c("center", "scale"))
```

```r
#Applying transformation
zScoreCont4 <- predict(preProcessCont4, as.data.frame(zScore))
head(zScoreCont4)
```

```
##        cont4
## 1 -0.2960488
## 2  0.7181924
## 3 -0.4988970
## 4  0.5153442
## 5 -0.4988970
## 6 -0.4988970
```

```r
# Summary statistics
summary(zScoreCont4)
```

```
##      cont4
##  Min.   :-0.4989
##  1st Qu.:-0.4989
##  Median :-0.4989
##  Mean   : 0.0000
##  3rd Qu.: 0.1096
##  Max.   :13.0919
```

Min-max normalization scales values to a fixed range (typically 0 to 1), ensuring that all values fall within a specific range. This is useful when preserving relative distances in the data.

```r
# Selecting cont5 for Min-Max normalization
minMax <- subset(bankdata, select = c(cont5))

#Preprocess using range method
preProcessCont5 <- preProcess(as.data.frame(minMax), method = c("range"))

#Applying transformation
minMaxCont5 <- predict(preProcessCont5, as.data.frame(minMax))

head(minMaxCont5)
```

```
##    cont5
## 1 0.1010
## 2 0.0215
## 3 0.1400
## 4 0.0500
## 5 0.0600
## 6 0.1800
```

```r
#Summary statistics
summary(minMaxCont5)
```

```
##      cont5
##  Min.   :0.00000
##  1st Qu.:0.03762
##  Median :0.08000
##  Mean   :0.09106
##  3rd Qu.:0.13550
##  Max.   :1.00000
```

Decimal scaling adjusts values by moving the decimal point based on the maximum absolute value of the feature. It ensures that all values fall within a reasonable range

```
# Selecting cont6 for Decimal Scaling
decimalScaling <- subset(bankdata, select = c(cont6))
max_abs_value <- max(abs(decimalScaling$cont6))
scale_factor <- 10^floor(log10(max_abs_value) + 1)
decimalScalingCont6 <- decimalScaling / scale_factor

head(decimalScalingCont6)
```

```
##      cont6
## 1 0.000000
## 2 0.000560
## 3 0.000824
## 4 0.000003
## 5 0.000000
## 6 0.000000
```

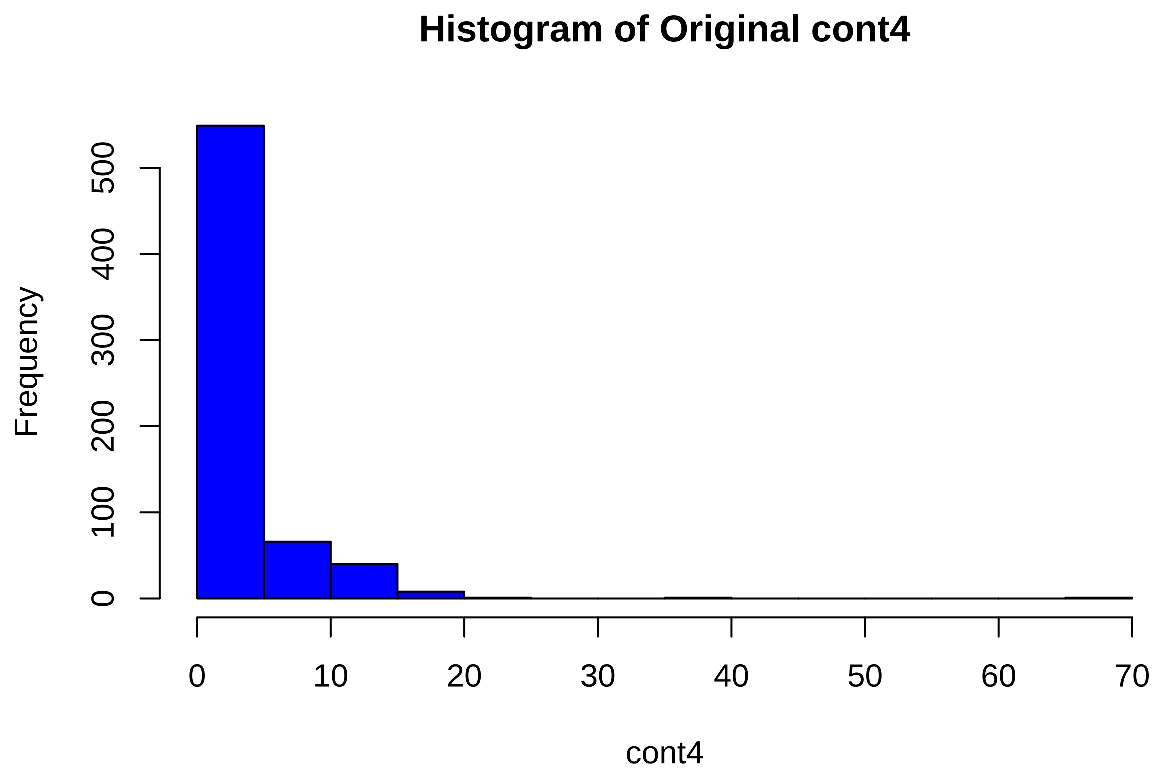
```
# Summary statistics
summary(decimalScalingCont6)
```

```
##      cont6
##  Min.   :0.0000000
##  1st Qu.:0.0000000
##  Median :0.0000050
##  Mean   :0.0009986
##  3rd Qu.:0.0003990
##  Max.   :0.1000000
```

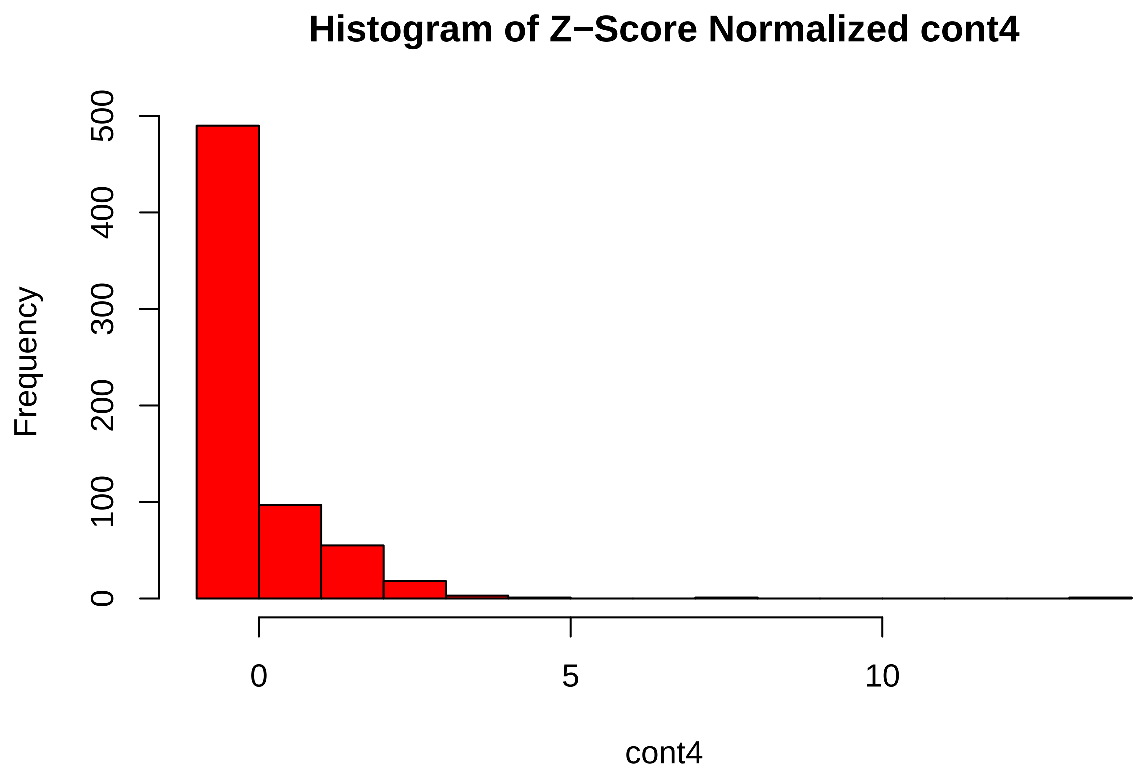
1.c Visualize the new distributions for the variables that have been normalized. What has changed from the previous visualization?

```
# Set layout to display 2 histograms per row
par(mfrow = c(3, 2))
```

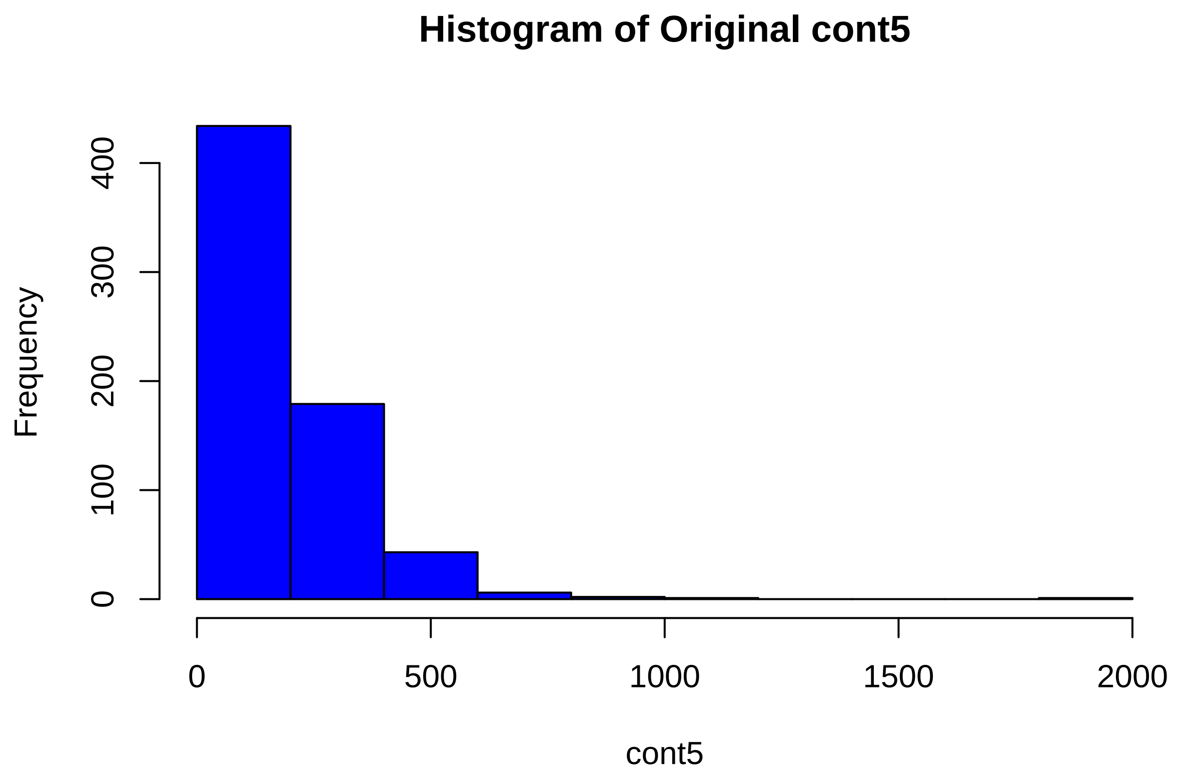
```
# Visualizations for cont4 before and after applying Z-score normalization
hist(bankdata$cont4, main = "Histogram of Original cont4", col = "blue", xlab = "cont4", border = "blac
```



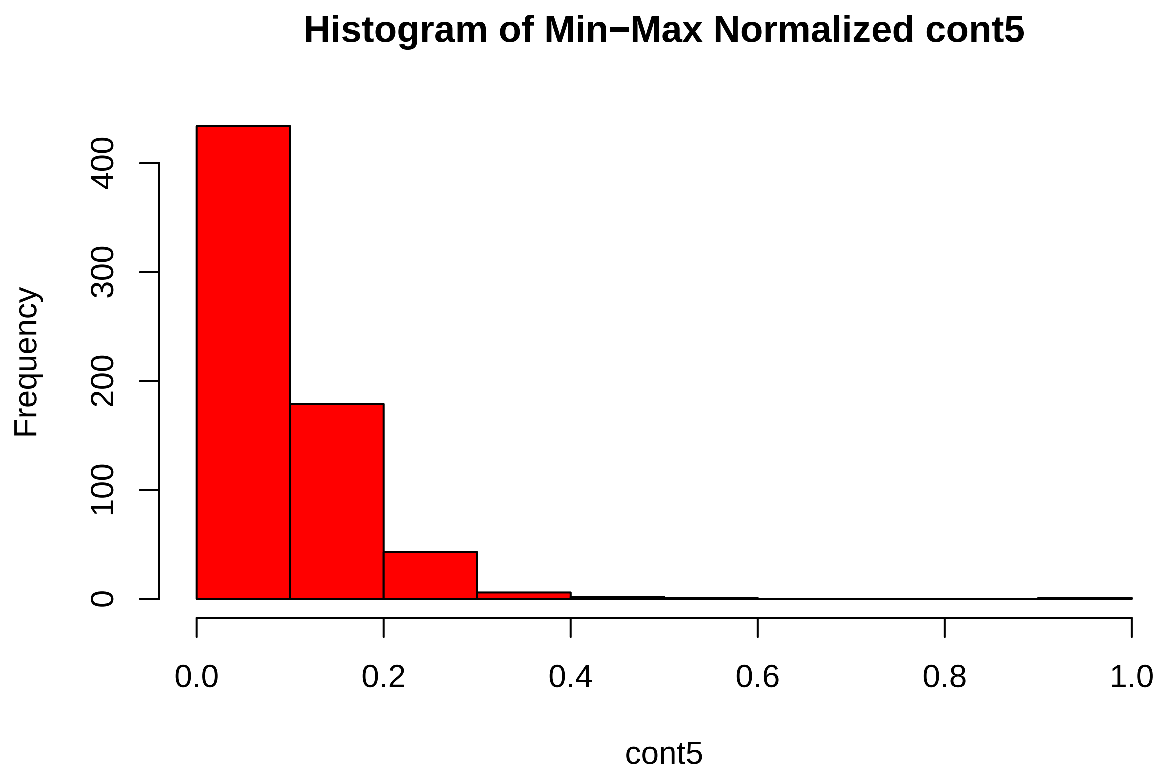
```
hist(zScoreCont4$cont4, main = "Histogram of Z-Score Normalized cont4", col = "red", xlab = "cont4", bo
```



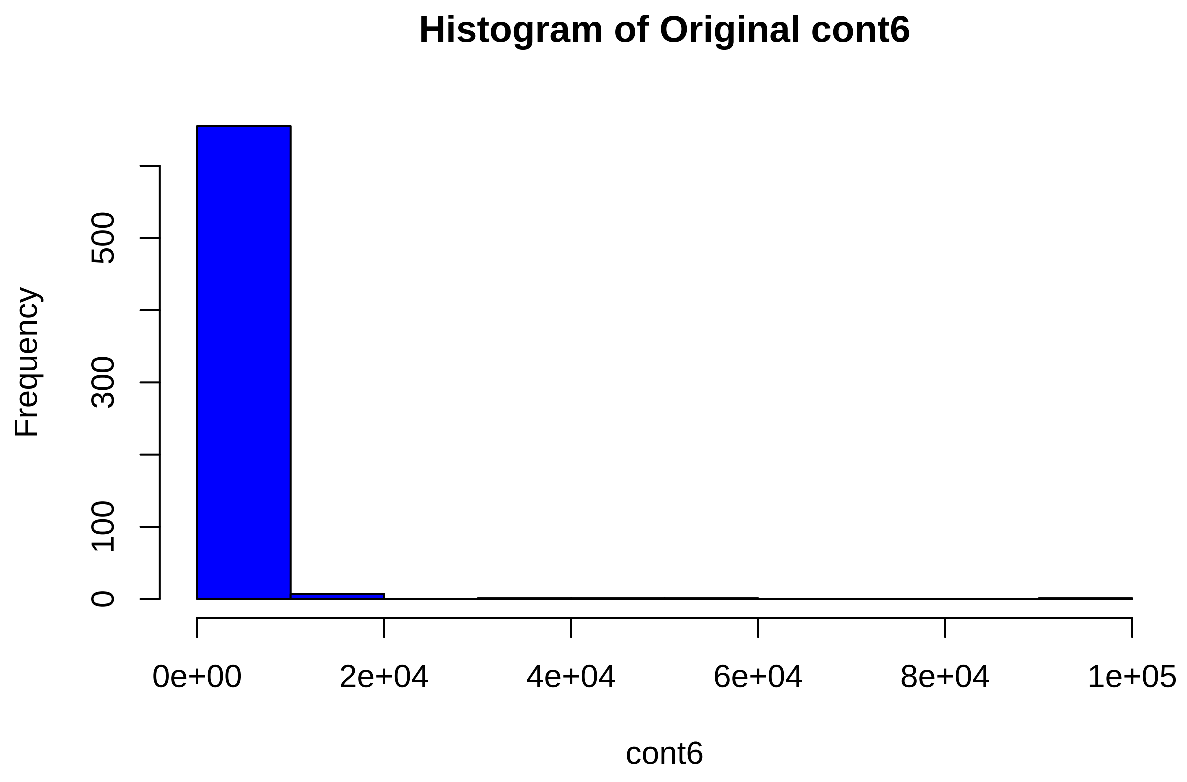
```
# Visualizations for cont5 before and after applying Min-Max normalization
hist(bankdata$cont5, main = "Histogram of Original cont5", col = "blue", xlab = "cont5", border = "blac
```



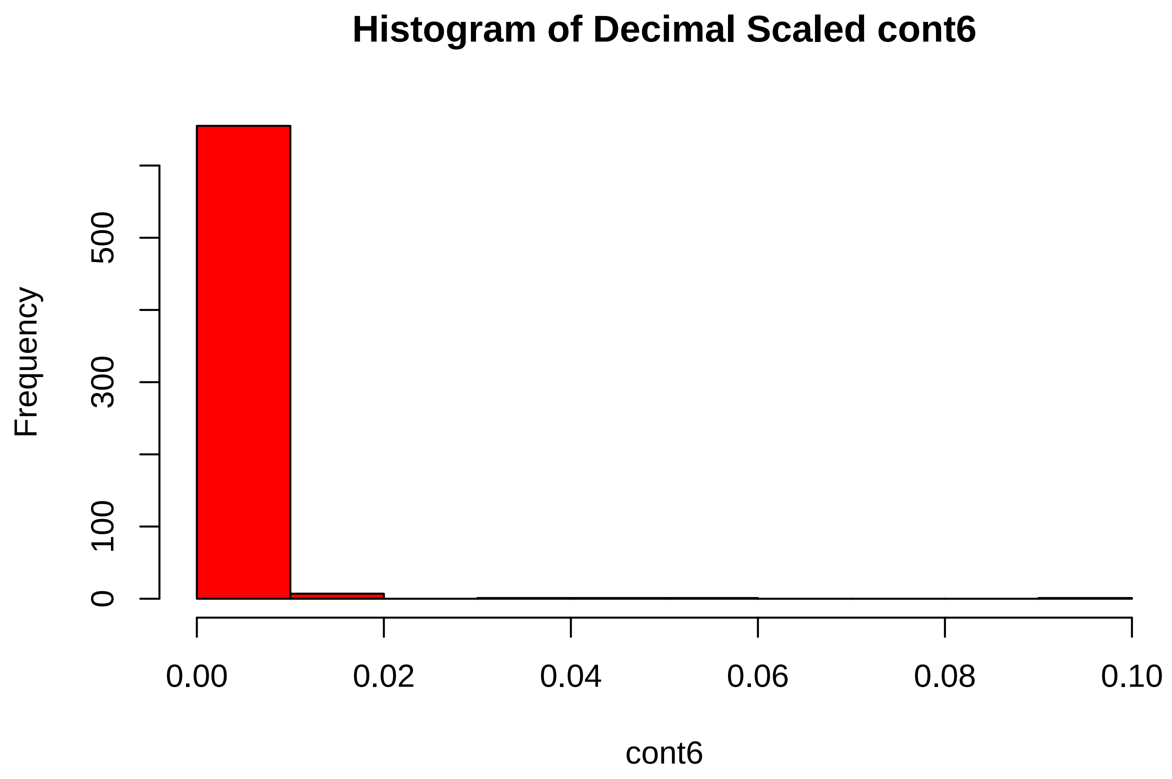
```r
hist(minMaxCont5$cont5, main = "Histogram of Min-Max Normalized cont5", col = "red", xlab = "cont5", bo
```



```
# Visualizations for cont6 before and after applying Decimal Scaling
hist(bankdata$cont6, main = "Histogram of Original cont6", col = "blue", xlab = "cont6", border = "blac
```



```r
hist(decimalScalingCont6$cont6, main = "Histogram of Decimal Scaled cont6", col = "red", xlab = "cont6"
```



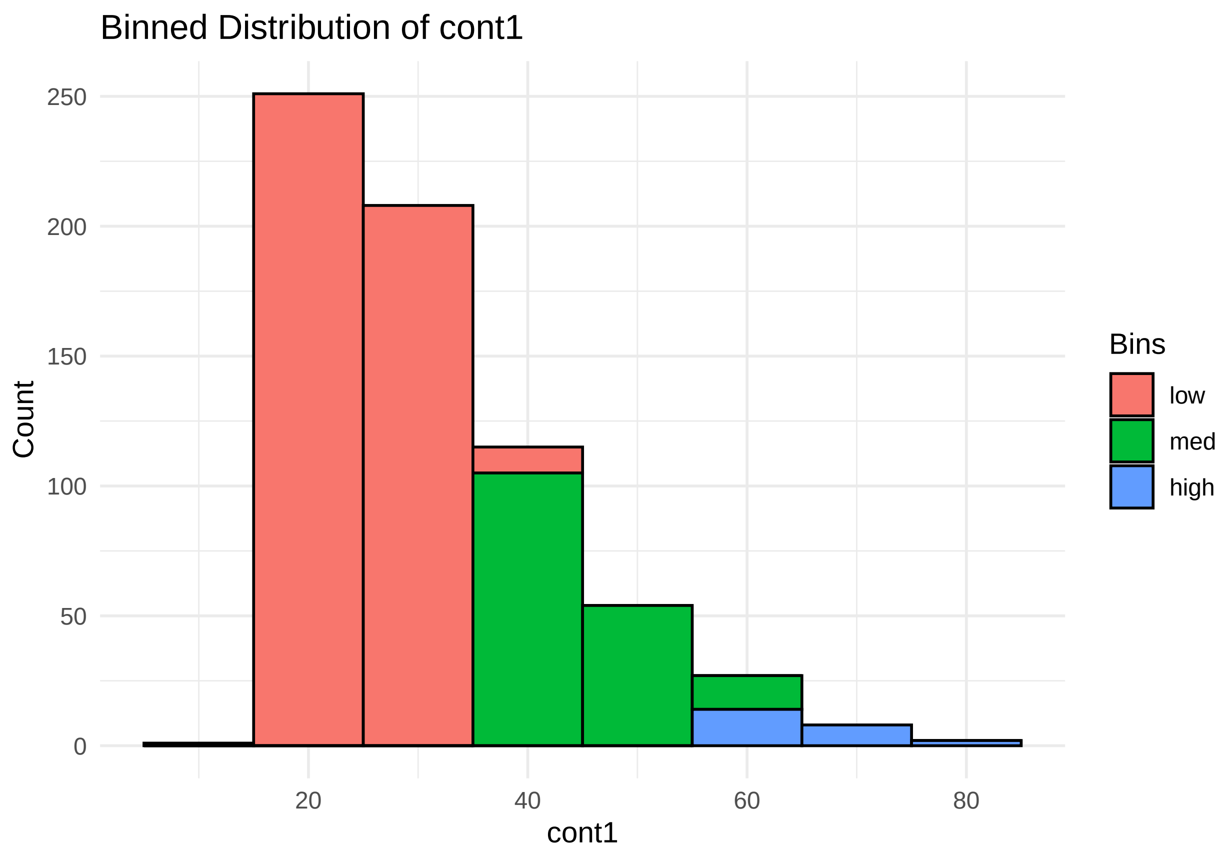
```r
# Reset layout to default
par(mfrow = c(1, 1))
```

1.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?

```r
library(ggplot2)
library(dplyr)

#Selecting cont1 as the variable to bin
vBankData <- bankdata %>% select(cont1)

# Convert into a dataframe
vBankData <- as.data.frame(vBankData)

# Apply binning (Equal-width bins)
vBankData <- vBankData %>% mutate(v_bins = cut(cont1, breaks = 3, labels = c("low", "med", "high")))

# Summary of binned variable
summary(vBankData)
```

```
##      cont1        v_bins
##  Min.   :13.75   low :470
##  1st Qu.:22.60   med :172
##  Median :28.50   high: 24
##  Mean   :31.57
##  3rd Qu.:38.25
##  Max.   :80.25
```

```r
# Visualizing the bins
ggplot(vBankData, aes(x = cont1, fill = v_bins)) +
  geom_histogram(binwidth = 10, color = "black") +
  labs(title = "Binned Distribution of cont1", x = "cont1", y = "Count", fill = "Bins") +
  theme_minimal()
```



Choice of Number of Values (3 bins - low, medium, high): Three bins provide a clear and simple categorization of the data without overcomplicating it.This makes it easy to interpret trends and compare groups.

Choice of Equal-Width Binning: Ensures each bin covers an equal range of values, making it a fair comparison between categories.Useful when the variable does not have extreme skewness or outliers and is relatively evenly spread.

Why These Particular Ranges? The bins are determined automatically based on the minimum and maximum values of cont1. Since the bins are equal-width, this method is unbiased and systematic, making it ideal for general analysis.

1.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.

```r
low <- vBankData %>% filter(v_bins == 'low') %>% mutate(meanValue = mean(cont1, na.rm = TRUE))

medium <- vBankData %>% filter(v_bins == 'med') %>% mutate(meanValue = mean(cont1, na.rm = TRUE))

high <- vBankData %>% filter(v_bins == 'high') %>% mutate(meanValue = mean(cont1, na.rm = TRUE))

vBankData <- bind_rows(low, medium, high)

head(vBankData)
```

```
##   cont1 v_bins meanValue
## 1 30.83    low  25.21155
## 2 24.50    low  25.21155
## 3 27.83    low  25.21155
## 4 20.17    low  25.21155
## 5 32.08    low  25.21155
## 6 33.17    low  25.21155
```

I used bin-based smoothing by replacing each value of cont1 in a bin with the mean of that bin. This reduces noise while preserving the overall distribution. It simplifies the data, making patterns clearer and easier to analyze.

2.a. 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.

```r
# Load necessary libraries
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.4.2
```

```r
library(caret)
library(dplyr)
```

```r
folds <- 10
accuracyReport <- createFolds(bankdata$approval, folds, returnTrain = T)
train_control <- trainControl(index = accuracyReport, method = 'cv', number=folds)
svm_model <- train(approval ~., data = bankdata, method = "svmLinear", trControl =
train_control)

svm_model
```

```
## Support Vector Machines with Linear Kernel
##
## 666 samples
##  11 predictor
##   2 classes: '-', '+'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 599, 600, 599, 599, 599, 601, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.863541  0.7289263
##
## Tuning parameter 'C' was held constant at a value of 1
```

2.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?

```
grid <- expand.grid(C = 10 ^ seq(-5, 2, 1))
gridSearch <- train(approval ~ ., data = bankdata, method = "svmLinear", trControl = train_control,tune

gridSearch
```

```
## Support Vector Machines with Linear Kernel
##
## 666 samples
##  11 predictor
##   2 classes: '-', '+'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 599, 600, 599, 599, 599, 601, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##   1e-05  0.5510427  0.0000000
##   1e-04  0.5510427  0.0000000
##   1e-03  0.8394548  0.6714881
##   1e-02  0.8635410  0.7289263
##   1e-01  0.8635410  0.7289263
##   1e+00  0.8635410  0.7289263
##   1e+01  0.8635410  0.7289263
##   1e+02  0.8635410  0.7289263
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

The chosen C parameter from the grid search was 0.01. The accuracy obtained with this optimal C value is 0.8634781 based on 10-fold cross-validation. This means that C = 0.01 provided the best trade-off between bias and variance, resulting in the highest model performance.

2.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 accuracy result for C = 1 can be different between (a) and (b) due to several reasons: 1. Randomness in Cross-Validation Splits- The way data is split into folds during 10-fold cross-validation might be different in each run, leading to slight variations in accuracy. Even if C = 1 was tested in both cases, the data partitions may not be exactly the same.

2. Hyperparameter Search Impact- In (b), grid search is used to optimize C. Since the model is trained with multiple C values, the overall training process could be slightly different compared to the default setting in (a), leading to different accuracy outcomes.

3.Numerical Stability & Optimization- The SVM optimization process can behave differently when searching across multiple values in grid search. The way the solver converges might be slightly different, leading to small variations in results.

4.Preprocessing Differences (Implicit or Explicit) - If there were any implicit preprocessing differences between runs (e.g., feature scaling, handling of class imbalances), these could also affect the final accuracy.

Key Takeaway: Even though C = 1 is included in both cases, the differences in cross-validation splits, search behavior, solver convergence, or preprocessing could cause slight variations in accuracy.

3.a.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.

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

```r
# Load required libraries
library(dplyr)
library(fastDummies)
```

```
## Warning: package 'fastDummies' was built under R version 4.4.2
```

```r
# Define columns to remove
dummy <- c("films", "vehicles", "starships", "name")

# Remove selected columns
res <- starwars[,!(names(starwars) %in% dummy)]

# Remove rows with missing values
res <- na.omit(res)

# Convert categorical variables to dummy variables
res <- dummy_cols(res, select_columns = c("hair_color", "skin_color", "eye_color", "sex", "homeworld",

head(res)
```

```
## # 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>, ‘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>,
## #   skin_color_brown <int>, ‘skin_color_brown mottle‘ <int>,
## #   skin_color_dark <int>, skin_color_fair <int>, skin_color_green <int>,
## #   skin_color_light <int>, skin_color_orange <int>, skin_color_pale <int>, ...
```

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

```r
# Load Required Libraries

library(dplyr)
library(caret)
library(fastDummies)
```

```r
# Prepare Data: Remove Unnecessary Columns
dummy <- c("films", "vehicles", "starships", "name")
res <- starwars[, !(names(starwars) %in% dummy)]

#Remove Missing Values
res <- na.omit(res)

# Convert Categorical Variables to Dummy Variables (Exclude gender)
res <- dummy_cols(res,
                  select_columns = c("hair_color", "skin_color", "eye_color", "homeworld", "species"),r

#Convert gender to factor (Target Variable)
res$gender <- as.factor(res$gender)

#Set up Cross-Validation
train_control <- trainControl(method = "cv", number = 10)

#Train SVM Model
svm_model <- train(gender ~ ., data = res, 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.
```

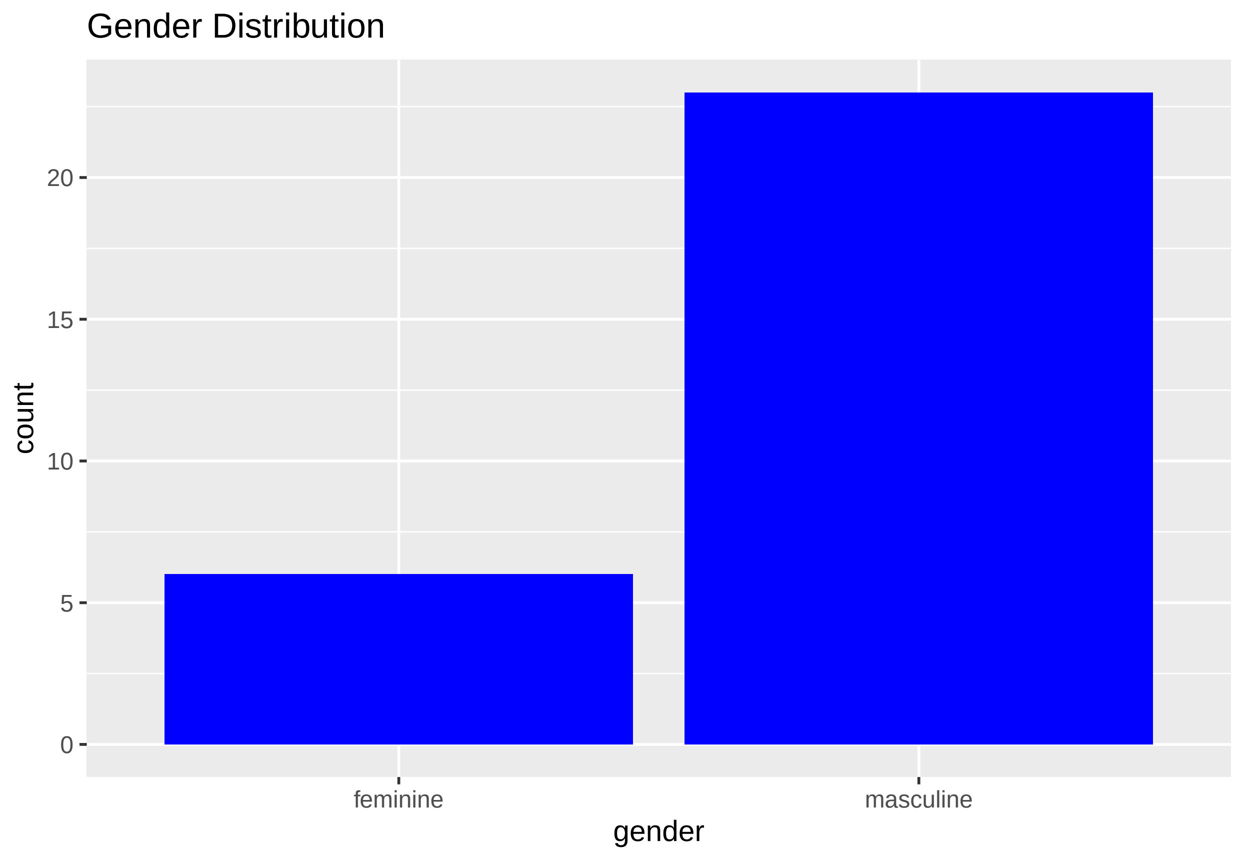
```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```r
svm_model
```

```
## Support Vector Machines with Linear Kernel
##
```

```
## 29 samples
## 60 predictors
##  2 classes: 'feminine', 'masculine'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 27, 26, 26, 26, 25, 27, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9083333  0.65
##
## Tuning parameter 'C' was held constant at a value of 1
```

```r
# Visualize Gender Distribution
ggplot(res, aes(gender)) + geom_bar(fill = "blue") + ggtitle("Gender Distribution")
```



3.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
# Load required libraries
library(dplyr)
library(caret)
library(ggplot2)
library(fastDummies)
```

```r
# Remove gender (target variable) before PCA
pca_data <- res %>% select(-gender)

# Convert categorical variables (factors) into numeric using dummy variables
pca_data <- dummy_cols(pca_data, remove_first_dummy = TRUE) # Create dummy variables
pca_data <- pca_data %>% select(where(is.numeric)) # Ensure only numeric columns remain

#Standardize and apply PCA
pre_process <- preProcess(pca_data, method = c("center", "scale", "pca"))
# Check variance explained by PCA components
summary(pre_process)
```

```
##                    Length Class  Mode
## dim                   2   -none- numeric
## bc                    0   -none- NULL
## yj                    0   -none- NULL
## et                    0   -none- NULL
## invHyperbolicSine     0   -none- NULL
## mean                 60   -none- numeric
## std                  60   -none- numeric
## ranges                0   -none- NULL
## rotation           1260   -none- numeric
## method                4   -none- list
## thresh                1   -none- numeric
## pcaComp               0   -none- NULL
## numComp               1   -none- numeric
## ica                   0   -none- NULL
## wildcards             2   -none- list
## k                     1   -none- numeric
## knnSummary            1   -none- function
## bagImp                0   -none- NULL
## median                0   -none- NULL
## data                  0   -none- NULL
## rangeBounds           2   -none- numeric
```

```r
# Transform the data using PCA
df.pc <- predict(pre_process, pca_data)

# Convert to dataframe and add gender back
df.pc <- as.data.frame(df.pc)
df.pc$gender <- res$gender

# Show first few rows of transformed dataset
head(df.pc)
```

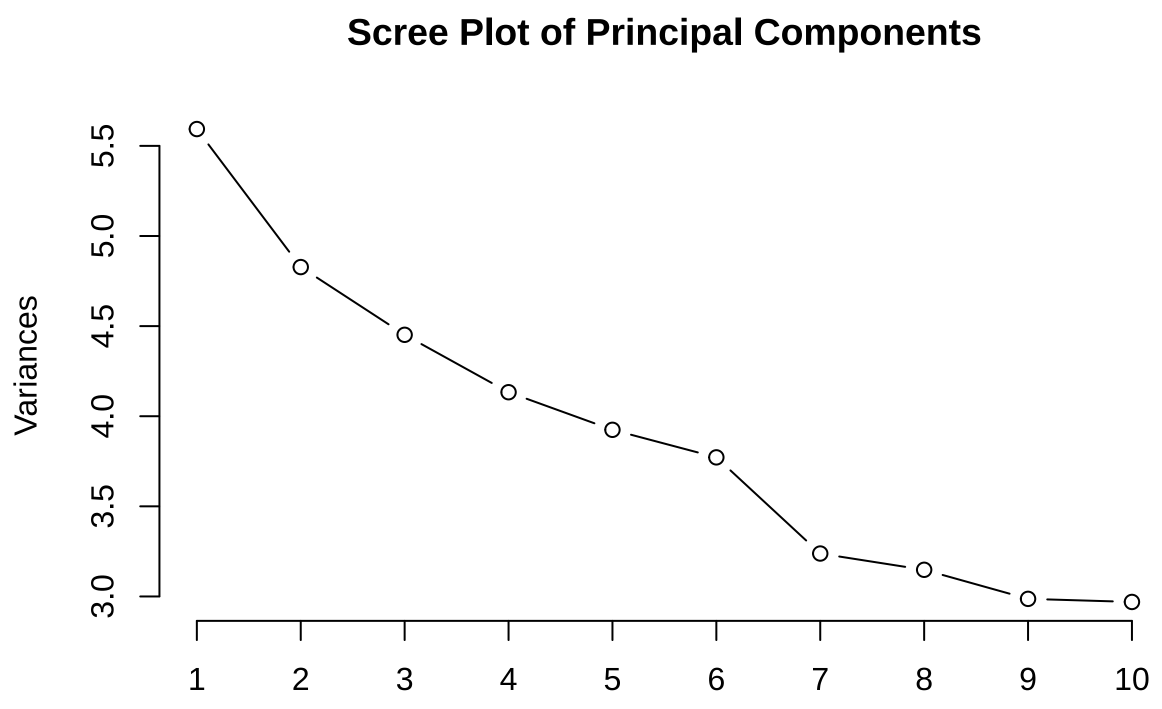
```
##          PC1         PC2        PC3         PC4        PC5        PC6
## 1 -0.6375360  1.43731181 -0.9058017  0.62427501 -1.3530477  0.6248423
## 2  2.4244035  0.65975357 -1.4005106 -0.07201041 -1.4380171 -1.0201507
## 3 -2.4168356 -0.05883147 -0.4836446  0.30359140  0.3279610  0.3795906
## 4  0.2595462  1.98482917 -0.5608812  0.70387199 -1.2478361  0.6893119
## 5 -1.3755270  1.25635723  0.2445083  0.38941980 -0.1764523  0.5729788
## 6 -1.0735696  0.89632861 -0.8837286  0.68797708 -1.2417584  0.3605301
##          PC7         PC8        PC9        PC10      PC11       PC12       PC13
## 1  1.4482835 -1.50187165 -0.2422030 -1.59322053  1.239096 -0.7316978 -0.3266496
## 2  0.9168691 -1.27041492  0.2393193 -1.01644347 -1.129906 -1.8956567 -0.2448952
## 3  0.6300629  0.02838024 -0.1068844 -0.06120313 -1.431425  1.1101240  0.1505802
## 4  1.6683951 -2.08903081  0.4982991 -2.15664636 -1.775054 -0.9315409 -0.4696252
## 5  1.7373627 -1.06442662 -0.1543680 -1.40614689 -1.548092  0.3992699 -0.1232273
## 6 -0.1297464 -0.66591174  0.7426064 -0.08125777 -1.238784 -0.1219051 -0.3893829
##         PC14      PC15        PC16        PC17       PC18        PC19
## 1 -0.01100399  2.211795  0.91816703  2.13187516 -0.7252977 -0.11556024
## 2 -1.77139013  1.974235 -0.68172257 -4.54174275  0.6091144 -1.08580748
## 3  1.15523024 -1.898991 -0.60684010 -0.34290619  0.4298352  0.06975323
## 4 -0.53114658 -1.085820  0.83079382  0.03904318  0.8689249  1.87236198
## 5  0.54118043 -1.223266 -0.14856615 -0.11237289  0.1378814  0.47296968
## 6  0.17479291 -0.490978  0.09992141 -0.25049622  1.3918612  0.28882864
##         PC20       PC21    gender
## 1 -0.51814395  1.2428617 masculine
## 2 -0.05896776  0.8146158 masculine
## 3 -0.73623432  1.4299556  feminine
## 4 -0.74845591 -3.5298254 masculine
## 5 -0.88068060  1.1959009  feminine
## 6 -0.10796004  0.4346187 masculine
```

```r
# Remove gender to prevent leakage
pca_data <- res %>% select(-gender)
```

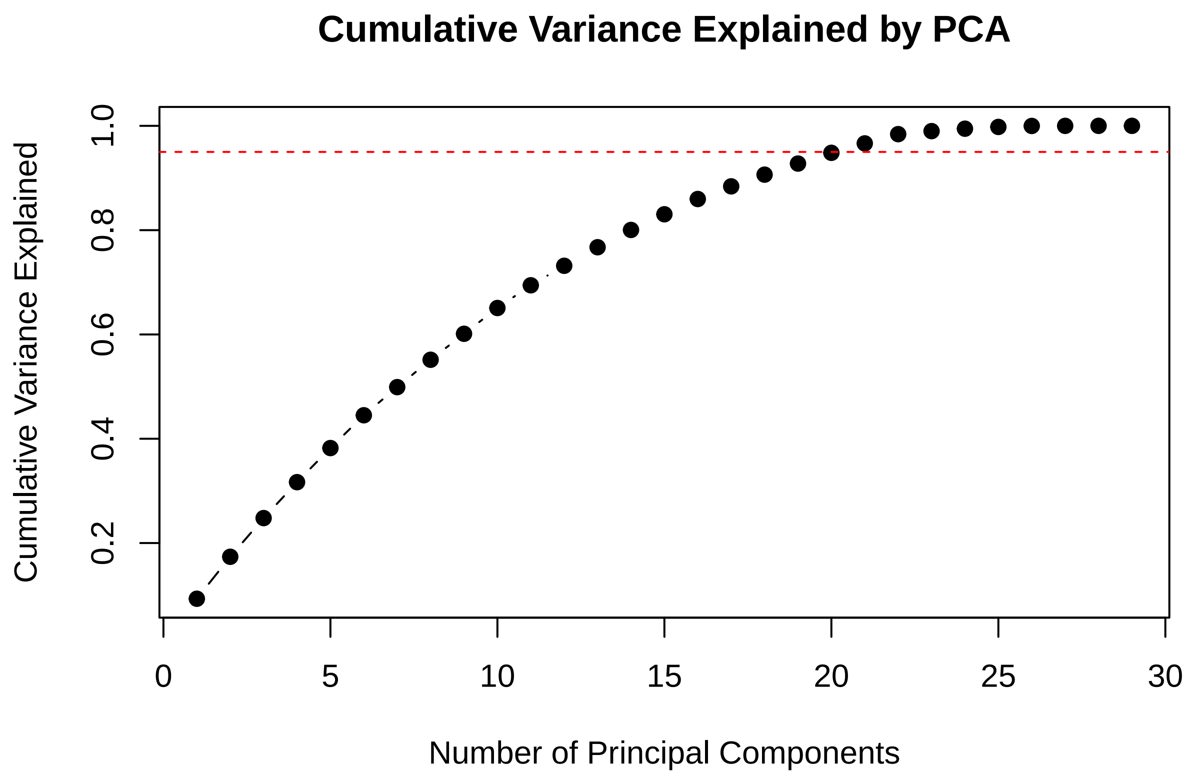
```r
# Convert categorical variables (factors) into numeric using dummy variables
pca_data <- dummy_cols(pca_data, remove_first_dummy = TRUE)
pca_data <- pca_data %>% select(where(is.numeric)) # Ensure only numeric columns remain

# Get PCA object using prcomp
pca_model <- prcomp(pca_data, center = TRUE, scale. = TRUE)
summary(pca_model)
```

```
## Importance of components:
##                            PC1     PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.36503 2.19719 2.1099 2.03303 1.98107 1.94211 1.79948
## Proportion of Variance 0.09322 0.08046 0.0742 0.06889 0.06541 0.06286 0.05397
## Cumulative Proportion  0.09322 0.17368 0.2479 0.31677 0.38218 0.44504 0.49901
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     1.77422 1.72818 1.72323 1.61597 1.49847 1.46072 1.41056
## Proportion of Variance 0.05246 0.04978 0.04949 0.04352 0.03742 0.03556 0.03316
## Cumulative Proportion  0.55147 0.60125 0.65074 0.69427 0.73169 0.76725 0.80041
##                           PC15    PC16    PC17    PC18    PC19    PC20   PC21
## Standard deviation     1.34230 1.32503 1.20667 1.16112 1.12901 1.11088 1.0422
## Proportion of Variance 0.03003 0.02926 0.02427 0.02247 0.02124 0.02057 0.0181
## Cumulative Proportion  0.83044 0.85970 0.88397 0.90644 0.92768 0.94825 0.9664
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     1.03195 0.58893 0.52832 0.44929 0.33895 0.10122 0.02936
## Proportion of Variance 0.01775 0.00578 0.00465 0.00336 0.00191 0.00017 0.00001
## Cumulative Proportion  0.98410 0.98988 0.99454 0.99790 0.99981 0.99999 1.00000
##                                PC29
## Standard deviation     8.577e-16
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

```
# Scree plot for principal components
screeplot(pca_model, type = "l", main = "Scree Plot of Principal Components")
```



```
# Compute cumulative variance explained
explained_variance <- cumsum(pca_model$sdev^2 / sum(pca_model$sdev^2))

#Plot cumulative variance explained
plot(explained_variance, type = "b", pch = 19,
     xlab = "Number of Principal Components",
     ylab = "Cumulative Variance Explained",
     main = "Cumulative Variance Explained by PCA")
abline(h = 0.95, col = "red", lty = 2)
```



```
# Choose the number of components that explain at least 95% variance
num_pcs <- which(explained_variance >= 0.95)[1]  # Find first component meeting threshold
cat("Selected number of principal components:", num_pcs, "\n")
```

```
## Selected number of principal components: 21
```

```
# Apply PCA transformation to keep only the selected components
pre_process <- preProcess(pca_data, method = "pca", pcaComp = num_pcs)
df_pc <- predict(pre_process, pca_data)

# Add gender back to the reduced dataset
df_pc <- as.data.frame(df_pc)
df_pc$gender <- res$gender

# Show the final dataset after PCA
head(df_pc)
```

```
##          PC1         PC2        PC3         PC4        PC5        PC6
## 1 -0.6375360  1.43731181 -0.9058017  0.62427501 -1.3530477  0.6248423
## 2  2.4244035  0.65975357 -1.4005106 -0.07201041 -1.4380171 -1.0201507
## 3 -2.4168356 -0.05883147 -0.4836446  0.30359140  0.3279610  0.3795906
## 4  0.2595462  1.98482917 -0.5608812  0.70387199 -1.2478361  0.6893119
## 5 -1.3755270  1.25635723  0.2445083  0.38941980 -0.1764523  0.5729788
## 6 -1.0735696  0.89632861 -0.8837286  0.68797708 -1.2417584  0.3605301
##          PC7         PC8        PC9        PC10       PC11        PC12        PC13
## 1  1.4482835 -1.50187165 -0.2422030 -1.59322053  1.239096 -0.7316978 -0.3266496
## 2  0.9168691 -1.27041492  0.2393193 -1.01644347 -1.129906 -1.8956567 -0.2448952
## 3  0.6300629  0.02838024 -0.1068844 -0.06120313 -1.431425  1.1101240  0.1505802
## 4  1.6683951 -2.08903081  0.4982991 -2.15664636 -1.775054 -0.9315409 -0.4696252
## 5  1.7373627 -1.06442662 -0.1543680 -1.40614689 -1.548092  0.3992699 -0.1232273
## 6 -0.1297464 -0.66591174  0.7426064 -0.08125777 -1.238784 -0.1219051 -0.3893829
##          PC14      PC15        PC16        PC17       PC18        PC19
## 1 -0.01100399  2.211795  0.91816703  2.13187516 -0.7252977 -0.11556024
## 2 -1.77139013  1.974235 -0.68172257 -4.54174275  0.6091144 -1.08580748
## 3  1.15523024 -1.898991 -0.60684010 -0.34290619  0.4298352  0.06975323
## 4 -0.53114658 -1.085820  0.83079382  0.03904318  0.8689249  1.87236198
## 5  0.54118043 -1.223266 -0.14856615 -0.11237289  0.1378814  0.47296968
## 6  0.17479291 -0.490978  0.09992141 -0.25049622  1.3918612  0.28882864
##          PC20       PC21    gender
## 1 -0.51814395  1.2428617 masculine
## 2 -0.05896776  0.8146158 masculine
## 3 -0.73623432  1.4299556  feminine
## 4 -0.74845591 -3.5298254 masculine
## 5 -0.88068060  1.1959009  feminine
## 6 -0.10796004  0.4346187 masculine
```

3.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.

```r
# Load necessary libraries
library(dplyr)
library(e1071)
library(caret)
library(tidyverse)
library(fastDummies)
library(caTools)
```

```
## Warning: package 'caTools' was built under R version 4.4.2
```

```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:purrr':
##
##     cross

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
# Load the Star Wars dataset and preprocess it
data(starwars, package = "dplyr")
result <- starwars[, !(names(starwars) %in% c("films", "vehicles", "starships", "name"))]
result <- na.omit(result)
result <- dummy_cols(result, select_columns = c("hair_color", "skin_color", "eye_color", "sex", "homewo
result$gender <- factor(result$gender)

#Split dataset into training and test sets
set.seed(1000)
split <- sample.split(result$gender, SplitRatio = 0.5)
training_set <- subset(result, split == TRUE)
test_set <- subset(result, split == FALSE)

#Perform PCA on numeric features
numeric_cols <- sapply(training_set, is.numeric)
train_numeric <- training_set[, numeric_cols]
train_numeric <- train_numeric[, apply(train_numeric, 2, var) > 0] # Remove zero variance columns

pca_model <- prcomp(train_numeric, center = TRUE, scale. = TRUE)
explained_variance <- cumsum(pca_model$sdev^2 / sum(pca_model$sdev^2))
num_components <- min(which(explained_variance >= 0.95)) # Select components covering 95% variance

# Apply PCA transformation to test set
test_numeric <- test_set[, numeric_cols]
test_numeric <- test_numeric[, names(train_numeric)]
train_pca <- as.data.frame(predict(pca_model, train_numeric))
test_pca <- as.data.frame(predict(pca_model, test_numeric))
train_pca <- train_pca[, 1:num_components]
test_pca <- test_pca[, 1:num_components]
train_pca$gender <- training_set$gender
test_pca$gender <- test_set$gender

#Train SVM model with grid search for optimal parameters
train_control <- trainControl(method = "cv", number = 10)
sigma_range <- sigest(gender ~ ., data = train_pca, frac = 1)
sigma_value <- as.numeric(sigma_range[1])
grid <- expand.grid(sigma = sigma_value, C = 2^(-5:5))

svm_model <- train(gender ~ ., data = train_pca, method = "svmRadial", trControl = train_control, tuneG
```

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

```r
#Best chosen parameters
best_C <- svm_model$bestTune$C
best_sigma <- svm_model$bestTune$sigma
print(paste("Best C parameter:", best_C))
```

```
## [1] "Best C parameter: 0.03125"
```

```r
print(paste("Best Sigma parameter:", best_sigma))
```

```
## [1] "Best Sigma parameter: 0.0357652001287554"
```

```r
#Predict on test set and compute accuracy
y_pred <- predict(svm_model, test_pca)
accuracy <- mean(y_pred == test_pca$gender)
print(paste("SVM Accuracy with PCA:", accuracy))
```

```
## [1] "SVM Accuracy with PCA: 0.785714285714286"
```

```r
#Generate confusion matrix
y_pred <- factor(y_pred, levels = levels(test_pca$gender))
test_pca$gender <- factor(test_pca$gender)
conf_matrix <- confusionMatrix(y_pred, test_pca$gender)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  feminine masculine
##   feminine         0         0
##   masculine        3        11
##
##                Accuracy : 0.7857
##                  95% CI : (0.492, 0.9534)
##     No Information Rate : 0.7857
##     P-Value [Acc > NIR] : 0.6483
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.7857
##              Prevalence : 0.2143
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : feminine
##
```

The accuracy is 78.57%.

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

PCA has significantly reduced the complexity of the model by lowering the number of features while retaining most of the important information. Initially, the dataset had many features, including dummy variables for categorical data, which could increase computational costs and the risk of overfitting. By applying PCA, only the most significant principal components that explain 95% of the variance were retained, effectively reducing the dimensionality of the data. This not only speeds up model training but also helps SVM perform better since high-dimensional spaces can negatively impact SVM efficiency. However, while PCA simplifies the model and makes training more efficient, it also results in a loss of interpretability, as the transformed principal components are combinations of multiple original features rather than distinct variables. Although

PCA has streamlined the model and made it computationally less expensive, it has not directly improved accuracy because the dataset suffers from class imbalance, which PCA alone cannot resolve.

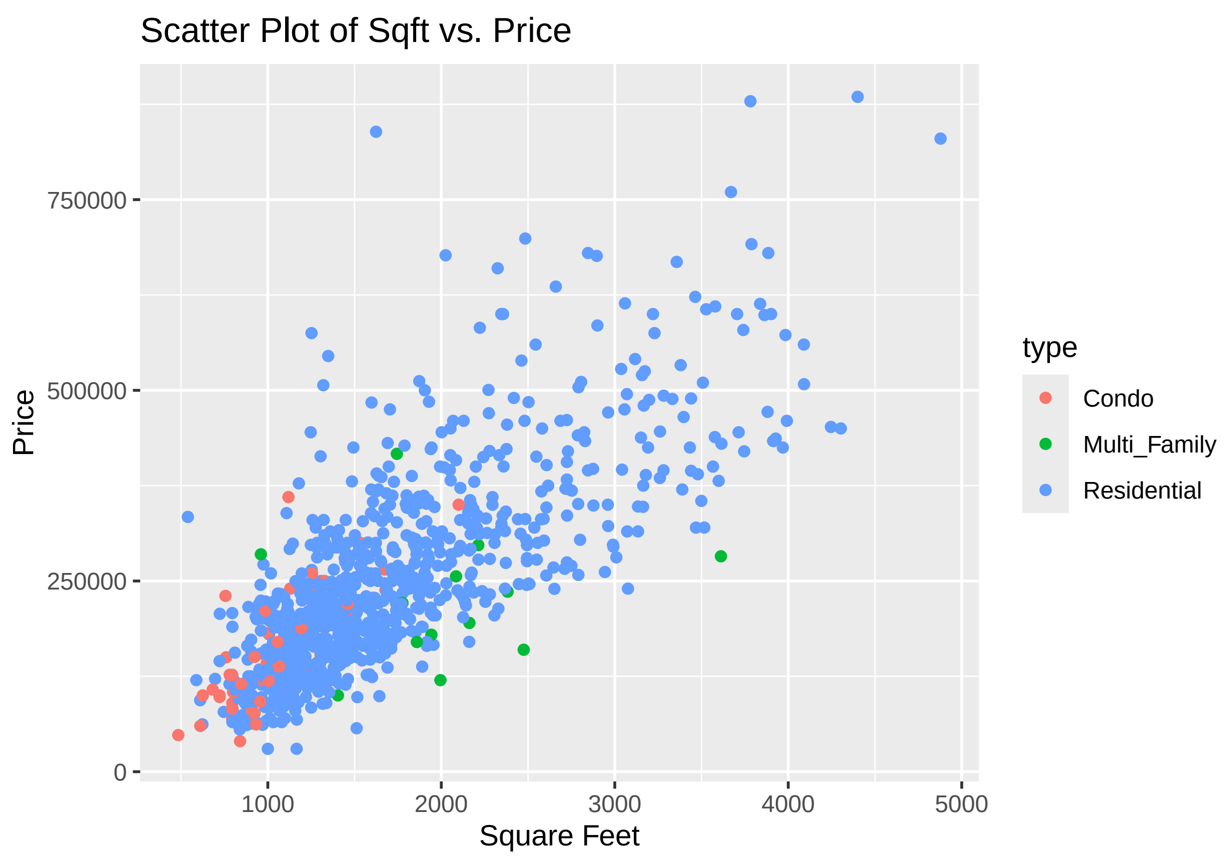
4.a.Use the Sacremento data from the caret library by running data(Sacremento) after loading caret. This data is about housing prices in Sacramento, California. Remove the zip and city variables.

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?

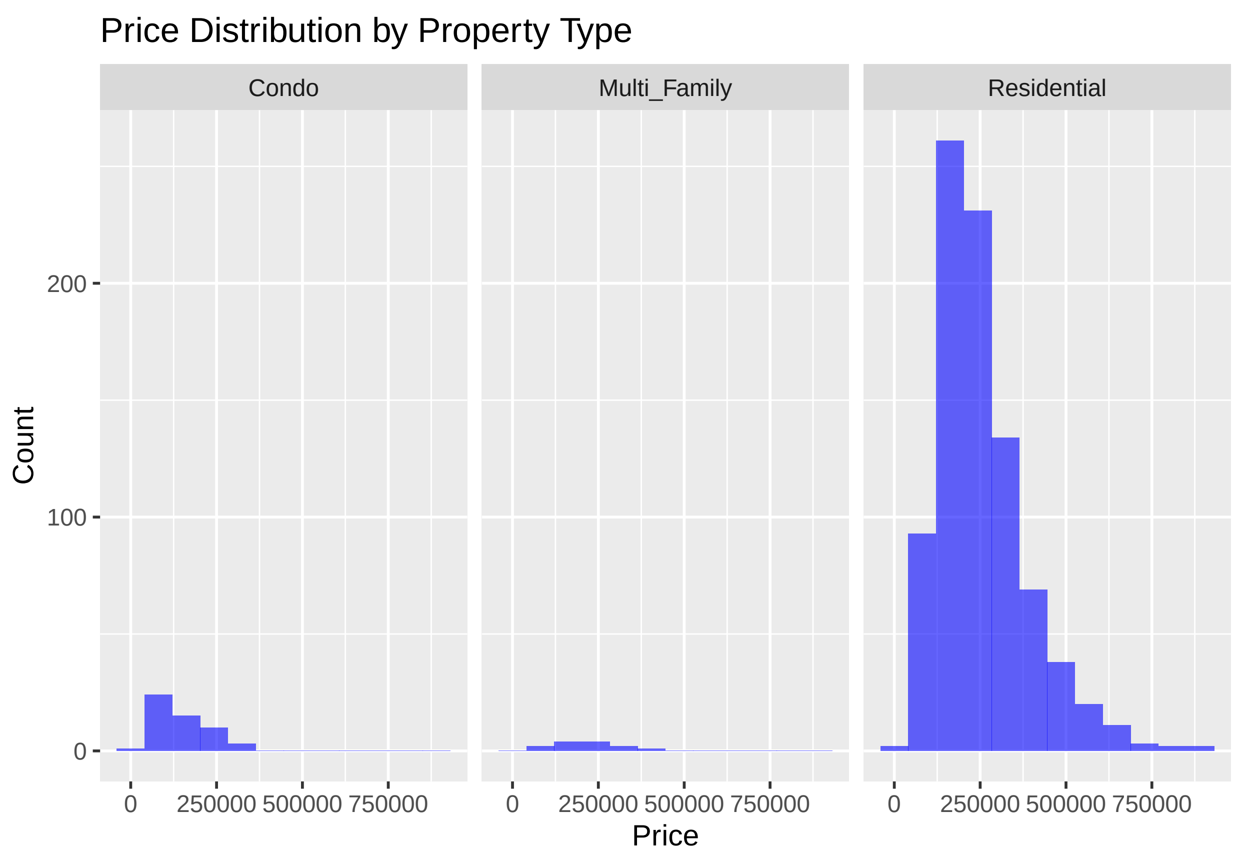
```r
# Load necessary libraries
library(caret)
library(dplyr)
library(ggplot2)
library(fastDummies)

#Load the Sacramento dataset
data("Sacramento")

#Remove unnecessary columns (zip, city, latitude, longitude)
sacData <- select(Sacramento, -c(zip, city, latitude, longitude))

# Check structure of dataset
str(sacData)
```

```
## 'data.frame':    932 obs. of  5 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 ...
```

```r
# Visualize relationship between sqft and price using scatter plot
ggplot(sacData, aes(x = sqft, y = price, color = type)) + geom_point() +
  labs(title = "Scatter Plot of Sqft vs. Price", x = "Square Feet", y = "Price")
```



```r
# Price distribution across property types
ggplot(sacData, aes(x = price)) +geom_histogram(binwidth = 81000, fill = "blue", alpha = 0.6) + facet_wi
labs(title = "Price Distribution by Property Type", x = "Price", y = "Count")
```



4.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 unnecessary columns
sac <- Sacramento %>% select(-c(city, zip, latitude, longitude))

# Apply log transformation to skewed numerical variables
sac <- sac %>%
mutate(log_price = log(price), log_sqft = log(sqft)) %>%
 select(-price, -sqft)  # Remove original price and sqft columns

# Set up 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Train SVM model using linear kernel with preprocessing (centering & scaling)
svmFold <- train(type ~ ., data = sac,
  method = "svmLinear", trControl = train_control, preProcess = c("center", "scale"))

svmFold
```

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

```
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 840, 839, 839, 838, 840, 837, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9303299  0.08642309
##
## Tuning parameter 'C' was held constant at a value of 1
```

4.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.

```
# Remove unnecessary columns (zip, city, latitude, longitude)
sac <- Sacramento %>% select(-c(city, zip, latitude, longitude))

# Apply log transformation to handle skewed variables
sac <- sac %>%
  mutate(log_price = log(price), log_sqft = log(sqft)) %>%
  select(-price, -sqft)  # Remove original price and sqft

# Set up 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define grid search parameters for SVM with radial kernel
svmGrid <- train(type ~ ., data = sac,
                 method = "svmRadial",
                trControl = train_control,
                 tuneGrid = expand.grid(C = c(0.01, 0.1, 1, 10), sigma = c(0.01, 0.1, 1, 10)), preProce

# Print best parameters and accuracy
best_C <- svmGrid$bestTune$C
best_sigma <- svmGrid$bestTune$sigma
accuracy <- max(svmGrid$results$Accuracy)
print(paste("Best C parameter:", best_C))
```

```
## [1] "Best C parameter: 10"
```

```
print(paste("Best Sigma parameter:", best_sigma))
```

```
## [1] "Best Sigma parameter: 0.1"
```

```
print(paste("Best SVM Accuracy:", accuracy))
```

```
## [1] "Best SVM Accuracy: 0.942160837996636"
```

```r
# Predict on training data
y_pred <- predict(svmGrid, sac)

# Generate confusion matrix
conf_matrix <- confusionMatrix(y_pred, sac$type)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction     Condo Multi_Family Residential
##   Condo           11            0           2
##   Multi_Family     0            6           0
##   Residential     42            7         864
##
## Overall Statistics
##
##                Accuracy : 0.9453
##                  95% CI : (0.9287, 0.959)
##     No Information Rate : 0.9292
##     P-Value [Acc > NIR] : 0.0286
##
##                   Kappa : 0.3843
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Condo Class: Multi_Family Class: Residential
## Sensitivity               0.20755            0.461538             0.9977
## Specificity               0.99772            1.000000             0.2576
## Pos Pred Value            0.84615            1.000000             0.9463
## Neg Pred Value            0.95430            0.992441             0.8947
## Prevalence                0.05687            0.013948             0.9292
## Detection Rate            0.01180            0.006438             0.9270
## Detection Prevalence      0.01395            0.006438             0.9796
## Balanced Accuracy         0.60264            0.730769             0.6276
```

```r
# Extract Kappa value
kappa_value <- conf_matrix$overall["Kappa"]
print(paste("Kappa Value:", kappa_value))
```

```
## [1] "Kappa Value: 0.384292542649515"
```

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

Since we previously applied log transformation to handle skewness in (b), here is a different approach: Removing outliers to improve the dataset before retraining the SVM model in (c).

```r
# Load necessary libraries
library(caret)
library(dplyr)
library(kernlab)

# Load Sacramento dataset
data("Sacramento")

# Remove unnecessary columns
sac <- Sacramento %>% select(-c(city, zip, latitude, longitude))

# Apply log transformation to skewed numerical variables
sac <- sac %>%
  mutate(log_price = log(price), log_sqft = log(sqft)) %>%
  select(-price, -sqft)  # Remove original price and sqft

# Function to remove outliers using IQR
remove_outliers <- function(df, column) {
  Q1 <- quantile(df[[column]], 0.25)
  Q3 <- quantile(df[[column]], 0.75)
  IQR_value <- Q3 - Q1

  df <- df %>% filter(df[[column]] >= (Q1 - 1.5 * IQR_value) & df[[column]] <= (Q3 + 1.5 * IQR_value))
  return(df)
}

# Apply outlier removal
sac <- remove_outliers(sac, "log_price")
sac <- remove_outliers(sac, "log_sqft")

# Check dataset size after outlier removal
print(dim(sac))  # Number of rows and columns after cleaning
```

```
## [1] 919   5
```

```r
# Set up 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define grid search parameters for SVM with radial kernel
svmGrid <- train(type ~ ., data = sac,
                 method = "svmRadial",
                 trControl = train_control,
                 tuneGrid = expand.grid(C = c(0.01, 0.1, 1, 10), sigma = c(0.01, 0.1, 1, 10)),
    preProcess = c("center", "scale"))

# Print best parameters and accuracy
best_C <- svmGrid$bestTune$C
best_sigma <- svmGrid$bestTune$sigma
accuracy <- max(svmGrid$results$Accuracy)
print(paste("Best C parameter:", best_C))
```

```
## [1] "Best C parameter: 10"
```

```r
print(paste("Best Sigma parameter:", best_sigma))
```

```
## [1] "Best Sigma parameter: 0.1"
```

```r
print(paste("Best SVM Accuracy:", accuracy))
```

```
## [1] "Best SVM Accuracy: 0.943530343161061"
```

```r
# Predict on training data
y_pred <- predict(svmGrid, sac)

# Generate confusion matrix
conf_matrix <- confusionMatrix(y_pred, sac$type)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction     Condo Multi_Family Residential
##   Condo           10            0           2
##   Multi_Family     0            6           0
##   Residential     41            7         853
##
## Overall Statistics
##
##                Accuracy : 0.9456
##                  95% CI : (0.9289, 0.9594)
##     No Information Rate : 0.9304
##     P-Value [Acc > NIR] : 0.03651
##
##                   Kappa : 0.375
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Condo Class: Multi_Family Class: Residential
## Sensitivity               0.19608            0.461538             0.9977
## Specificity               0.99770            1.000000             0.2500
## Pos Pred Value            0.83333            1.000000             0.9467
## Neg Pred Value            0.95480            0.992333             0.8889
## Prevalence                0.05550            0.014146             0.9304
## Detection Rate            0.01088            0.006529             0.9282
## Detection Prevalence      0.01306            0.006529             0.9804
## Balanced Accuracy         0.59689            0.730769             0.6238
```

```r
# Extract Kappa value
kappa_value <- conf_matrix$overall["Kappa"]
print(paste("Kappa Value:", kappa_value))
```

```
## [1] "Kappa Value: 0.374965993797268"
```

4.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. 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.

```r
# Load necessary libraries
library(caret)
library(dplyr)
library(ggplot2)

# Load Sacramento dataset
data("Sacramento")

# Remove unnecessary columns
sac <- Sacramento %>% select(-c(city, zip, latitude, longitude))

# Apply log transformation to handle skewed variables
sac <- sac %>%
  mutate(log_price = log(price), log_sqft = log(sqft)) %>%
  select(-price, -sqft)  # Remove original price and sqft

# Check class distribution before balancing
table(sac$type)
```

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

```r
# Identify majority and minority classes
class_counts <- table(sac$type)
majority_class <- names(which.max(class_counts))  # Largest class
minority_classes <- names(which(class_counts < max(class_counts)))  # Other classes

# Keep all minority class data
sac_minority <- sac %>% filter(type %in% minority_classes)

# Randomly sample from the majority class to match minority counts
set.seed(1000)  # Ensure reproducibility
sample_size <- nrow(sac_minority)  # Match sample size of minority classes
sac_majority_sampled <- sac %>% filter(type == majority_class) %>% sample_n(sample_size)

# Combine minority and sampled majority data
sac_balanced <- bind_rows(sac_minority, sac_majority_sampled)

# Check new class distribution
table(sac_balanced$type)
```

```
##
```

```
##        Condo Multi_Family  Residential
##           53           13           66
```

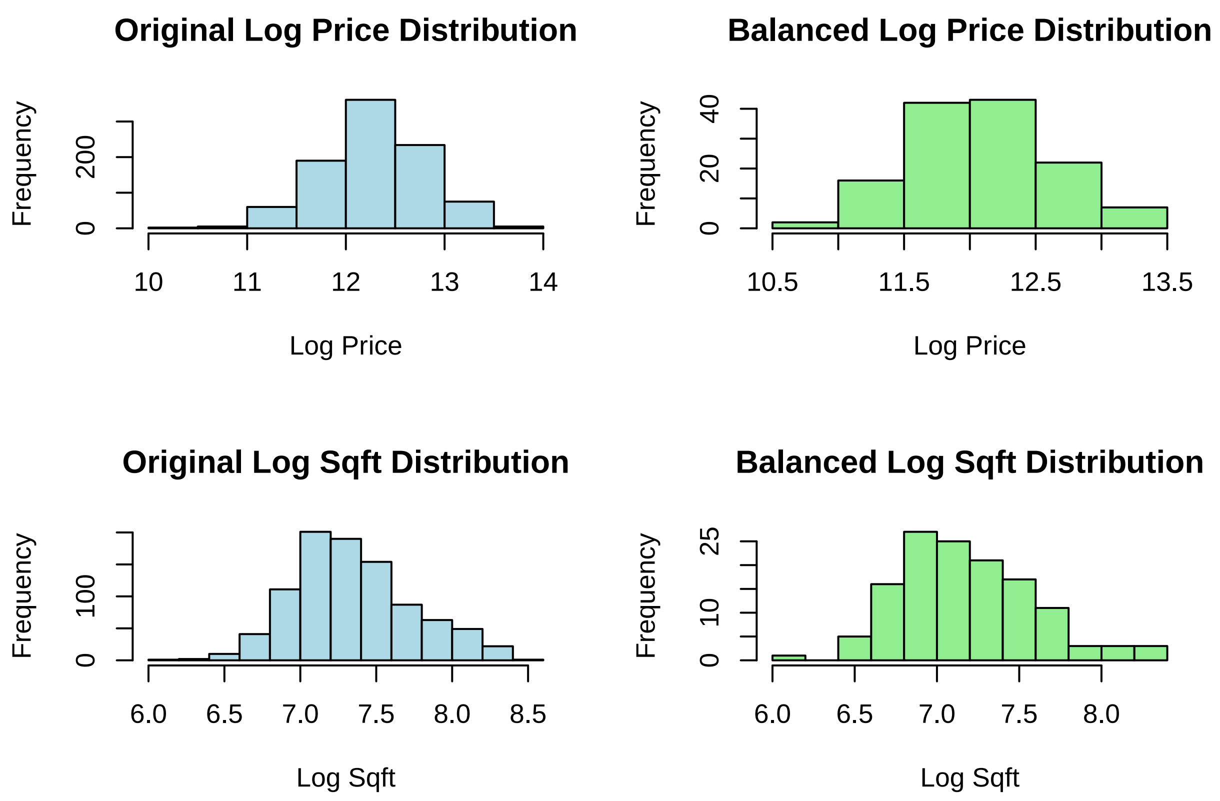
```r
# Summary statistics before and after balancing
summary(sac)
```

```
##       beds           baths                   type       log_price
##  Min.   :1.000   Min.   :1.000   Condo        : 53   Min.   :10.31
##  1st Qu.:3.000   1st Qu.:2.000   Multi_Family: 13   1st Qu.:11.96
##  Median :3.000   Median :2.000   Residential :866   Median :12.30
##  Mean   :3.276   Mean   :2.053                      Mean   :12.28
##  3rd Qu.:4.000   3rd Qu.:2.000                      3rd Qu.:12.63
##  Max.   :8.000   Max.   :5.000                      Max.   :13.69
##     log_sqft
##  Min.   :6.182
##  1st Qu.:7.062
##  Median :7.293
##  Mean   :7.346
##  3rd Qu.:7.577
##  Max.   :8.492
```

```r
summary(sac_balanced)
```

```
##       beds           baths                   type      log_price
##  Min.   :1.000   Min.   :1.000   Condo        :53   Min.   :10.60
##  1st Qu.:2.000   1st Qu.:1.000   Multi_Family:13   1st Qu.:11.71
##  Median :3.000   Median :2.000   Residential :66   Median :12.11
##  Mean   :2.894   Mean   :1.958                     Mean   :12.09
##  3rd Qu.:4.000   3rd Qu.:2.000                     3rd Qu.:12.44
##  Max.   :8.000   Max.   :4.000                     Max.   :13.46
##     log_sqft
##  Min.   :6.182
##  1st Qu.:6.885
##  Median :7.138
##  Mean   :7.181
##  3rd Qu.:7.468
##  Max.   :8.354
```

```r
# Visualize distributions to ensure balance
par(mfrow = c(2, 2))

hist(sac$log_price, main = "Original Log Price Distribution", col = "lightblue", xlab = "Log Price")

hist(sac_balanced$log_price, main = "Balanced Log Price Distribution", col = "lightgreen", xlab = "Log 

hist(sac$log_sqft, main = "Original Log Sqft Distribution", col = "lightblue", xlab = "Log Sqft")

hist(sac_balanced$log_sqft, main = "Balanced Log Sqft Distribution", col = "lightgreen", xlab = "Log Sq
```



```r
# Set up 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define grid search parameters for SVM with radial kernel
svmGrid <- train(type ~ ., data = sac_balanced,
                 method = "svmRadial",
                 trControl = train_control,
                 tuneGrid = expand.grid(C = c(0.01, 0.1, 1, 10), sigma = c(0.01, 0.1, 1, 10)),preProces

# Print best parameters and accuracy
best_C <- svmGrid$bestTune$C
best_sigma <- svmGrid$bestTune$sigma
accuracy <- max(svmGrid$results$Accuracy)
print(paste("Best C parameter:", best_C))
```

```
## [1] "Best C parameter: 10"
```

```r
print(paste("Best Sigma parameter:", best_sigma))
```

```
## [1] "Best Sigma parameter: 0.1"
```

```r
print(paste("Best SVM Accuracy on Balanced Data:", accuracy))
```

```
## [1] "Best SVM Accuracy on Balanced Data: 0.809249084249084"
```

```r
# Predict on training data
y_pred <- predict(svmGrid, sac_balanced)

# Generate confusion matrix
conf_matrix <- confusionMatrix(y_pred, sac_balanced$type)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction     Condo Multi_Family Residential
##   Condo           46            1           8
##   Multi_Family     0            7           0
##   Residential      7            5          58
##
## Overall Statistics
##
##                Accuracy : 0.8409
##                  95% CI : (0.7672, 0.8987)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 2.798e-16
##
##                   Kappa : 0.7171
##
##  Mcnemar's Test P-Value : 0.1084
##
## Statistics by Class:
##
##                      Class: Condo Class: Multi_Family Class: Residential
## Sensitivity                0.8679             0.53846             0.8788
## Specificity                0.8861             1.00000             0.8182
## Pos Pred Value             0.8364             1.00000             0.8286
## Neg Pred Value             0.9091             0.95200             0.8710
## Prevalence                 0.4015             0.09848             0.5000
## Detection Rate             0.3485             0.05303             0.4394
## Detection Prevalence       0.4167             0.05303             0.5303
## Balanced Accuracy          0.8770             0.76923             0.8485
```

```r
# Extract Kappa value
kappa_value <- conf_matrix$overall["Kappa"]
print(paste("Kappa Value:", kappa_value))
```

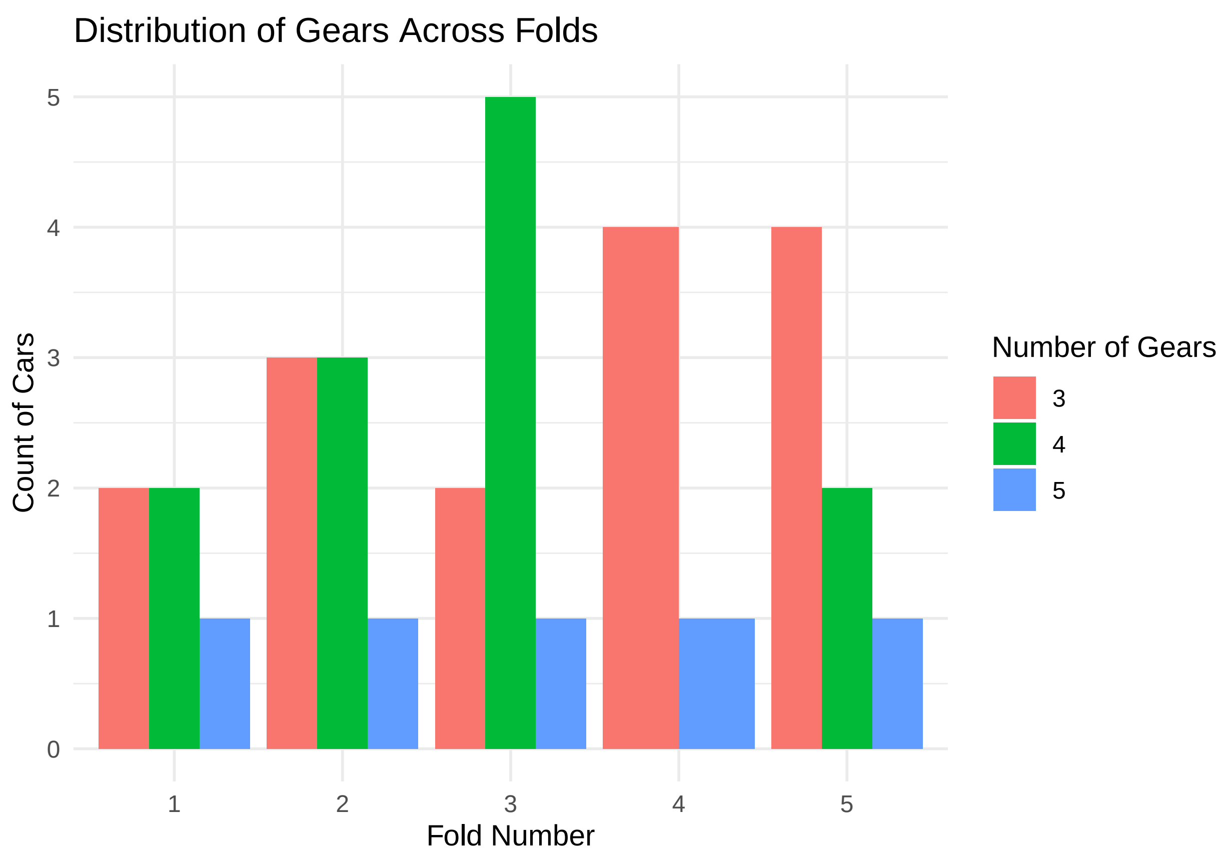
```
## [1] "Kappa Value: 0.717085119412125"
```

5.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.

```r
library(caret)
library(dplyr)
library(ggplot2)

# Create a copy of the mtcars dataset
mycars <- mtcars
unique(mycars$gear)
```

```
## [1] 4 3 5
```

```r
# Initialize a new variable for fold indices
mycars$folds <- 0

# Create 5 folds
set.seed(123)  # Ensures reproducibility
flds <- createFolds(1:nrow(mycars), k = 5, list = TRUE)

# Assign fold indices
for (i in 1:5) {
  mycars$folds[flds[[i]]] <- i
}

# Convert folds to a factor for correct visualization
mycars$folds <- factor(mycars$folds)

# Visualize the distribution of the 'gear' variable across folds
ggplot(mycars, aes(x = folds, fill = as.factor(gear))) +
  geom_bar(position = "dodge") +
  labs(title = "Distribution of Gears Across Folds",
       x = "Fold Number", y = "Count of Cars",
       fill = "Number of Gears") +
  theme_minimal()
```



The bar plot shows how the gear variable (3, 4 and 5 gears) is distributed across the 5-fold partitions of the mtcars dataset. The variation across folds indicates that while some folds contain a more balanced mix of gears, others may have more instances of a specific gear type, highlighting potential differences in data distribution across folds.