HW2

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

```
# Load necessary libraries
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.3.3
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: lattice
```

```r
# Set working directory (Update with your actual path if needed)
setwd("C:/Users/SAM/OneDrive/Documents/data science/as2")

# Load the dataset
df <- read_csv("BankData.csv")
```

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

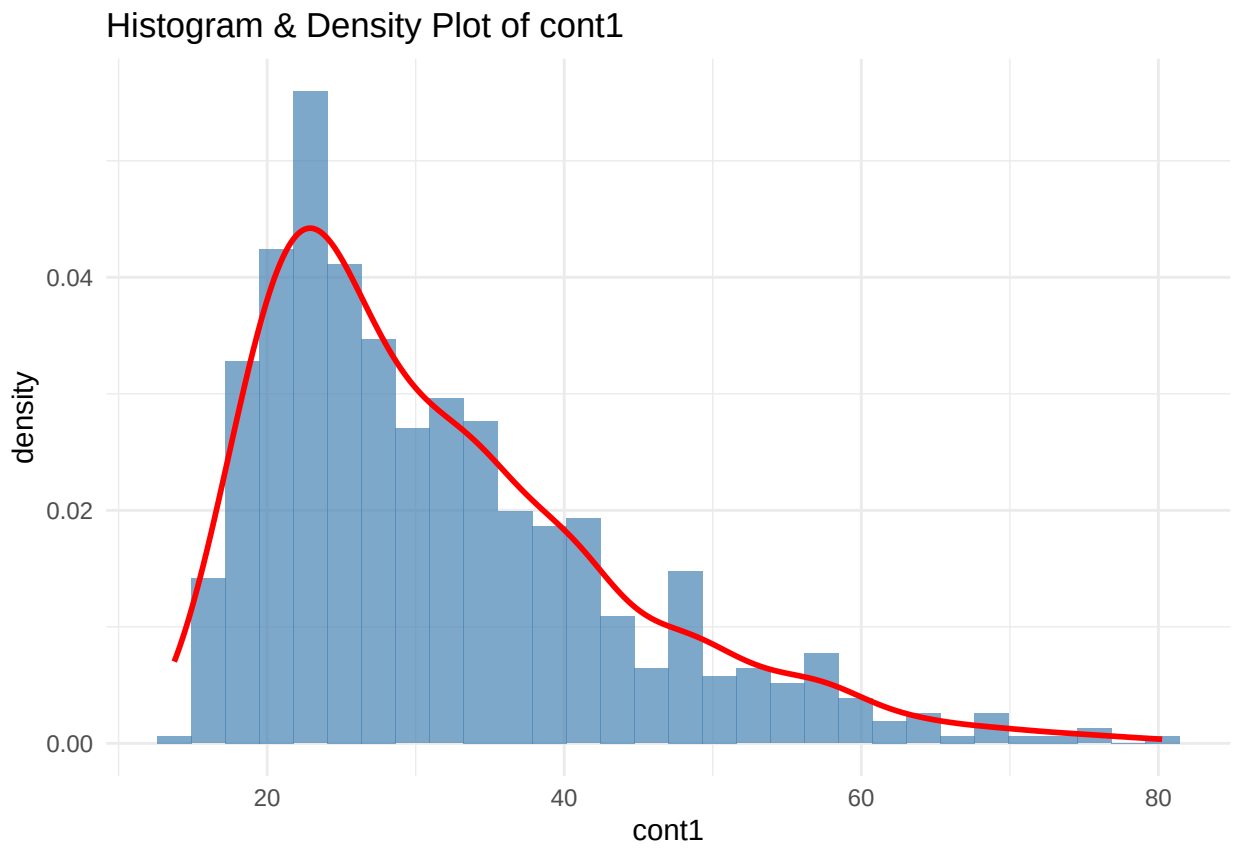
```r
# Remove the unnecessary index column
df <- df %>% select(-1)  # Removes the first column

# Convert Boolean-like variables ('t' and 'f') to factors
bool_cols <- c("bool1", "bool2", "bool3", "approval")  # Boolean-like categorical columns
df[bool_cols] <- lapply(df[bool_cols], as.factor)
```

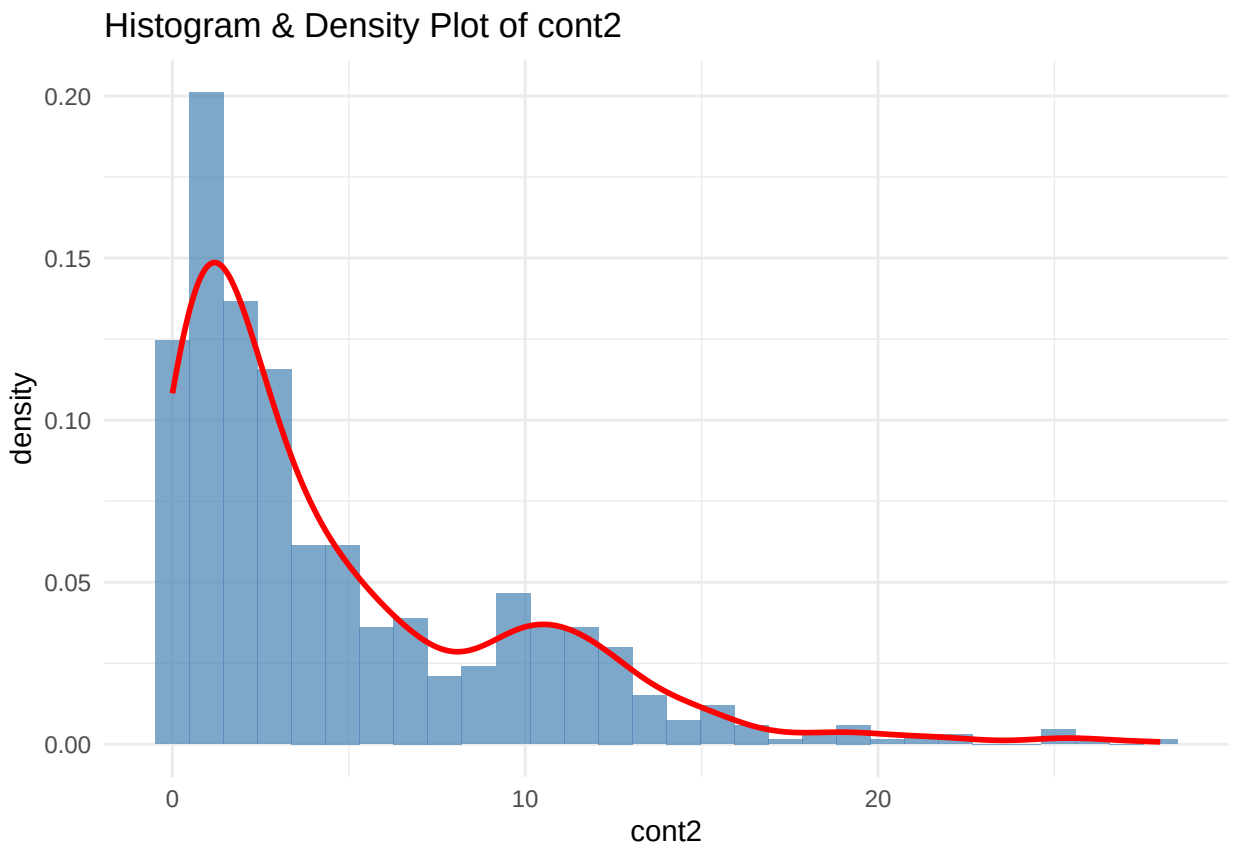
```r
# Question 1a: Visualizing the distributions of variables
# Loop through each column and generate appropriate plots
for (col in colnames(df)) {
  if (is.numeric(df[[col]])) {
    # Histogram with density plot for numeric variables
    p <- ggplot(df, aes(x = .data[[col]])) +
      geom_histogram(aes(y = ..density..), bins = 30, fill = "steelblue", alpha = 0.7) +
      geom_density(color = "red", size = 1) +
      ggtitle(paste("Histogram & Density Plot of", col)) +
      theme_minimal()
  } else if (is.factor(df[[col]])) {
    # Bar plot for categorical variables
    p <- ggplot(df, aes(x = .data[[col]])) +
      geom_bar(fill = "skyblue") +
      ggtitle(paste("Bar Plot of", col)) +
      theme_minimal()
  }
  print(p)  # Display the plot
}
```

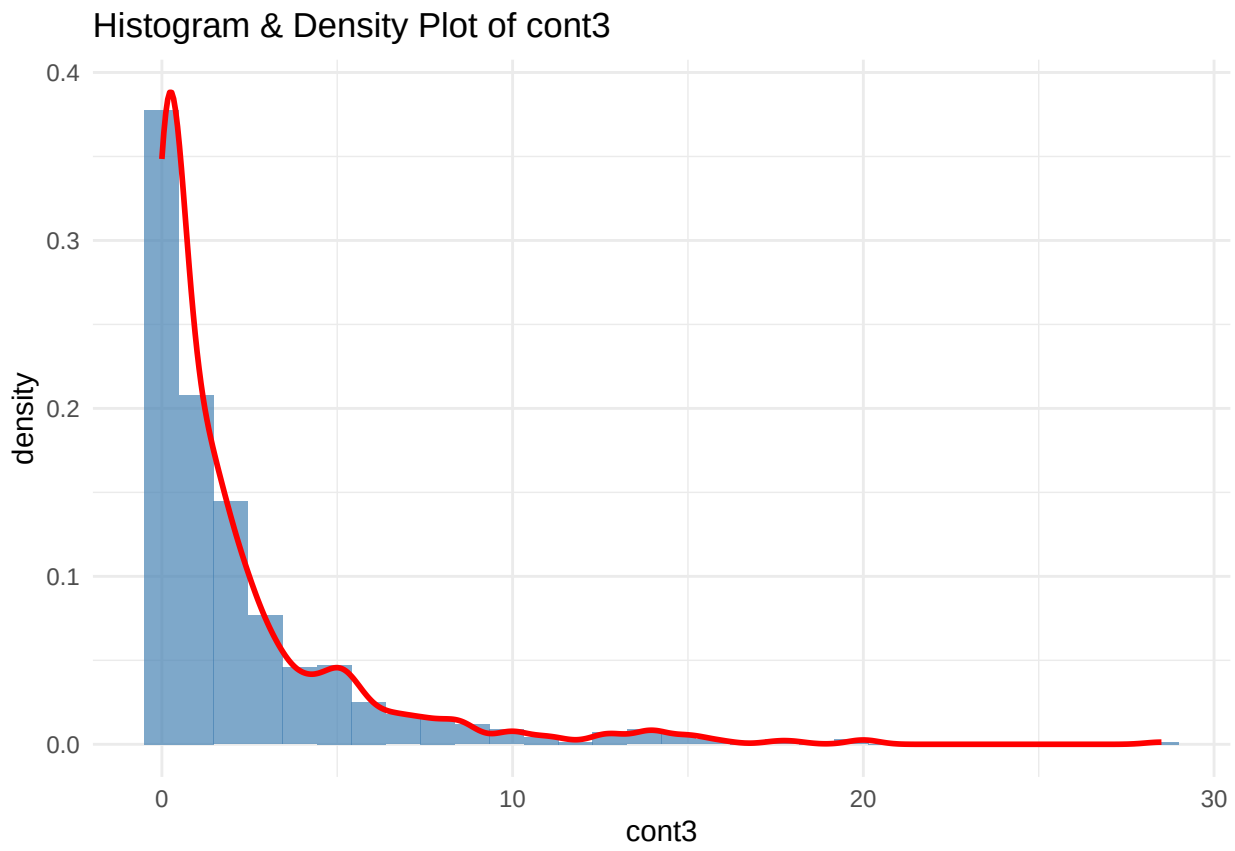
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

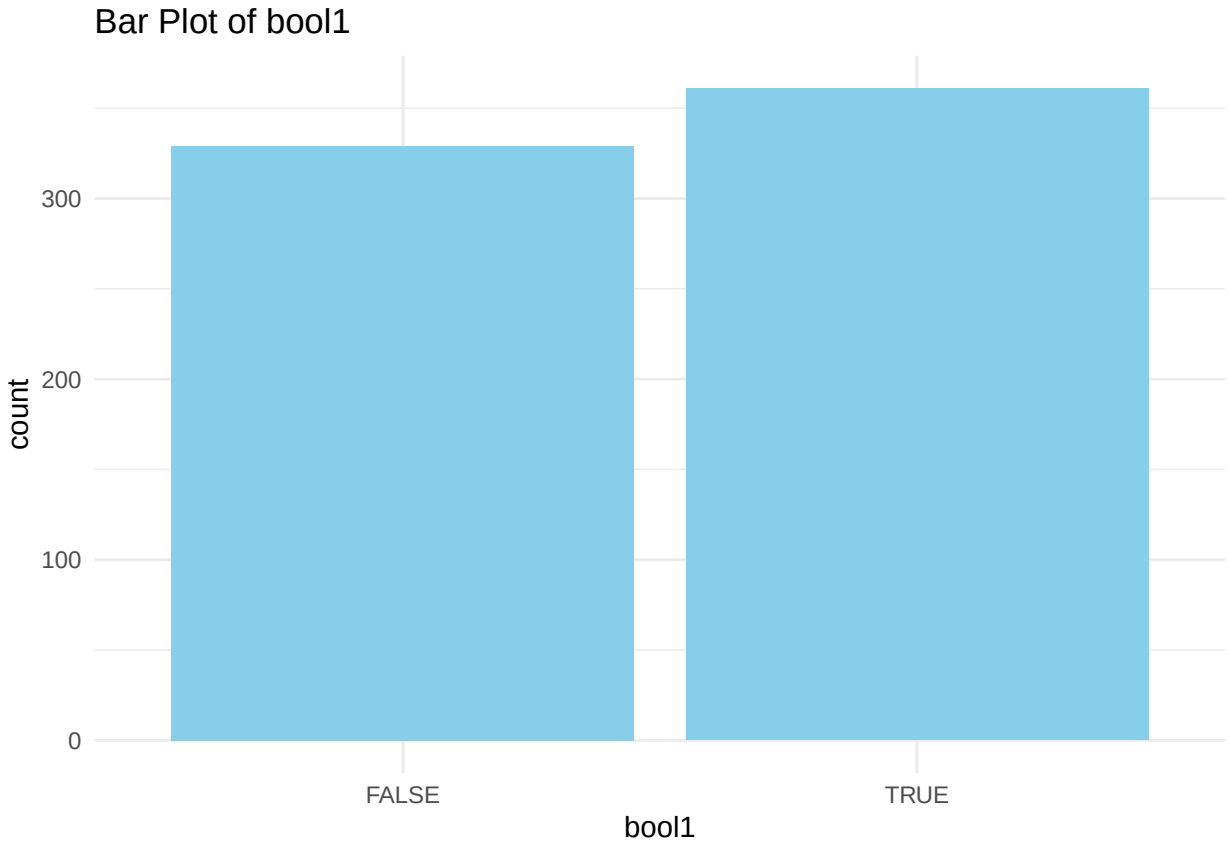
```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
```

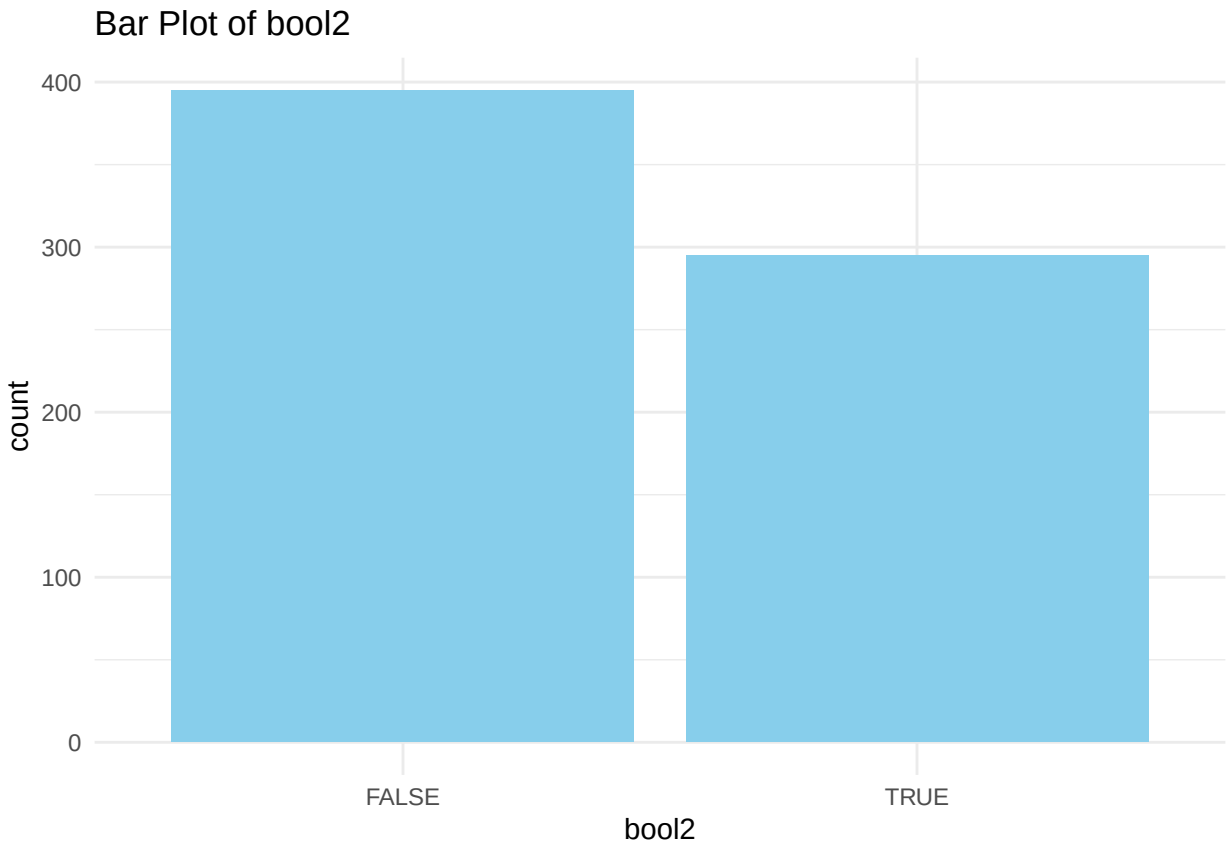
```
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.

## Warning: Removed 12 rows containing non-finite outside the scale range
## (‘stat_bin()‘).

## Warning: Removed 12 rows containing non-finite outside the scale range
## (‘stat_density()‘).
```

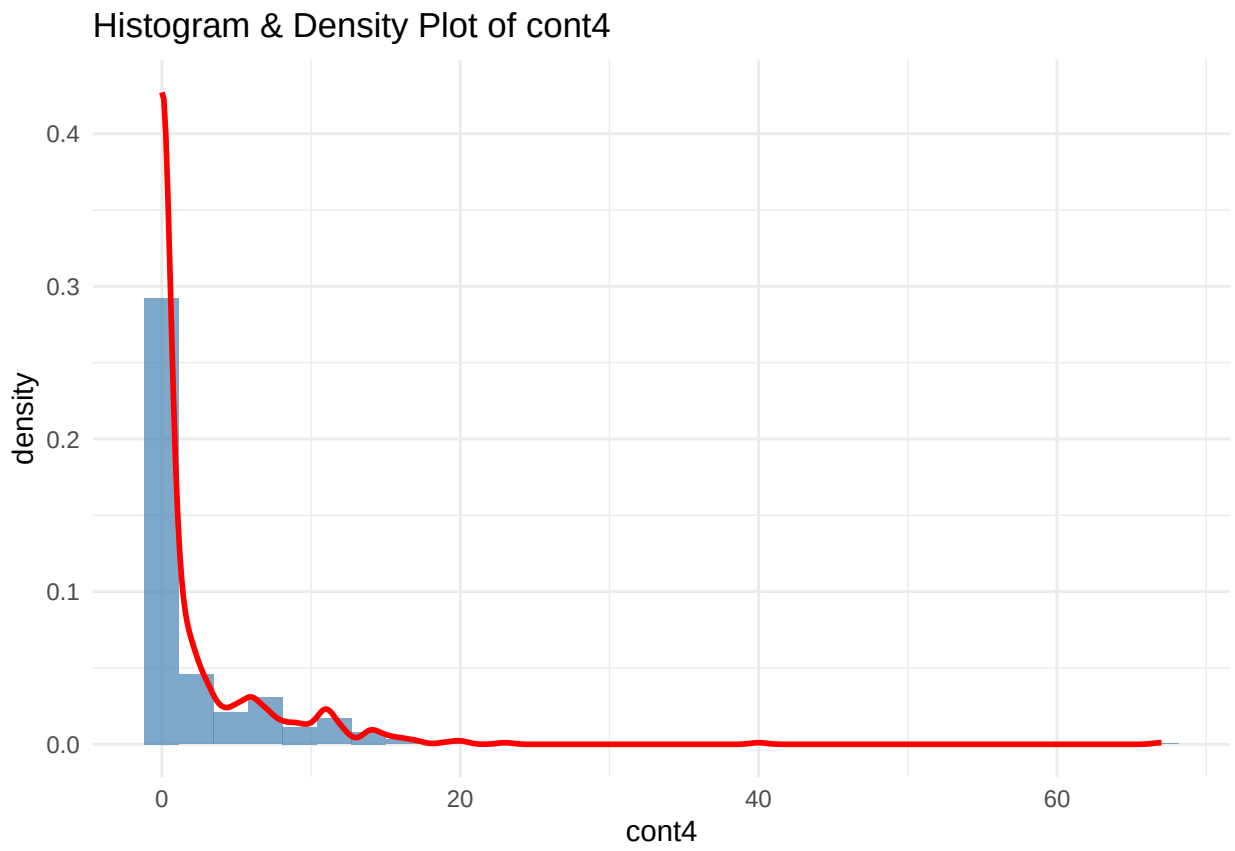


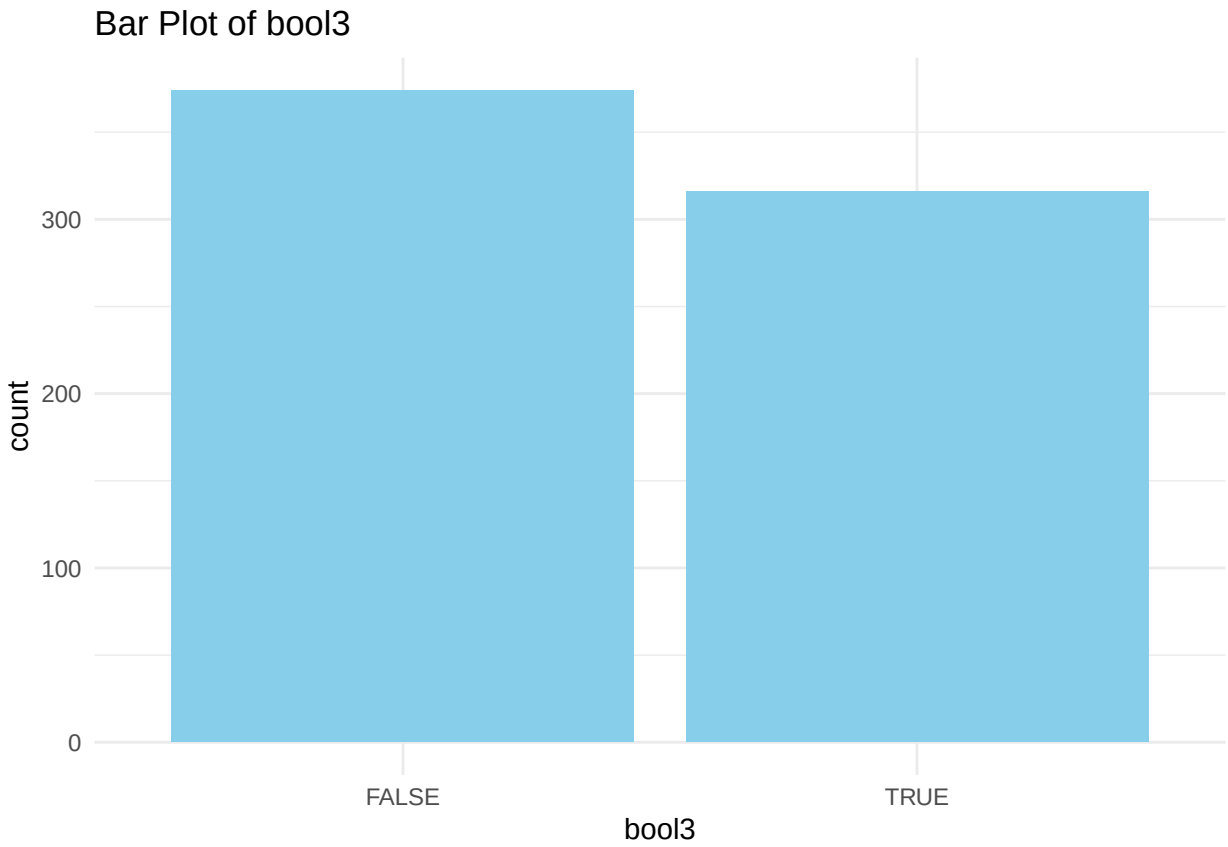

Histogram & Density Plot of cont2
density
0.20
0.15
0.10
0.05
0.00
0
10
20
cont2


Histogram & Density Plot of cont3
density
0.4
0.3
0.2
0.1
0.0
0
10
20
30
cont3

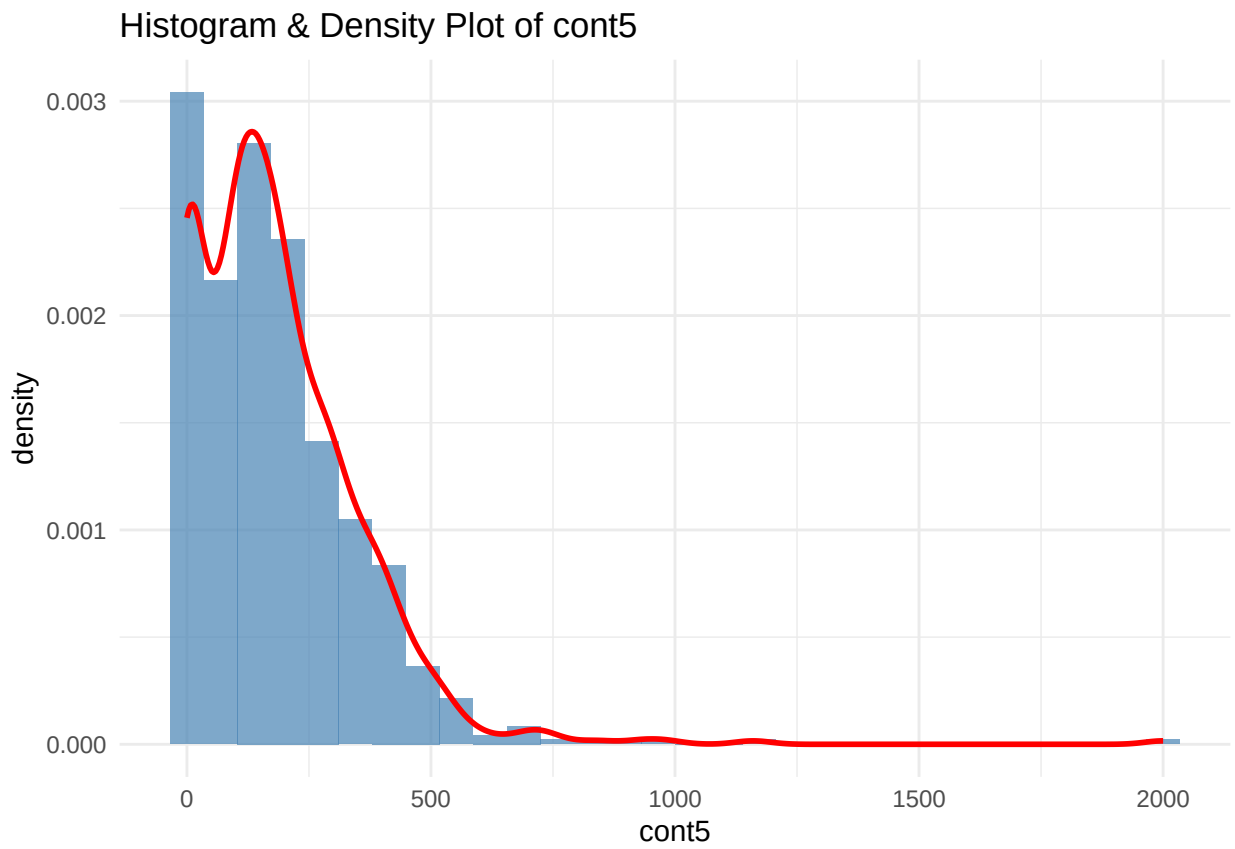

Bar Plot of bool1
300
200
100
0
count
FALSE
TRUE
bool1

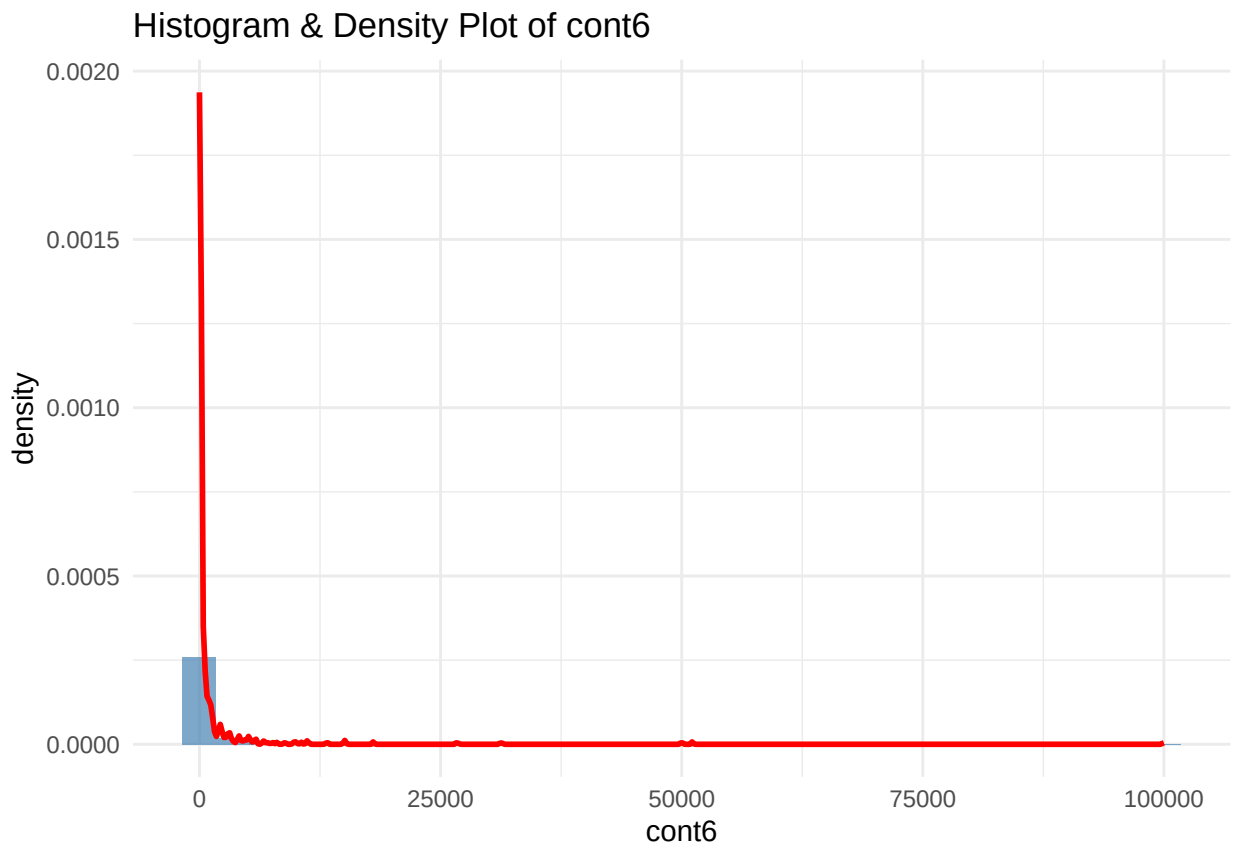

Bar Plot of bool2
400
300
200
100
0
count
FALSE
TRUE
bool2

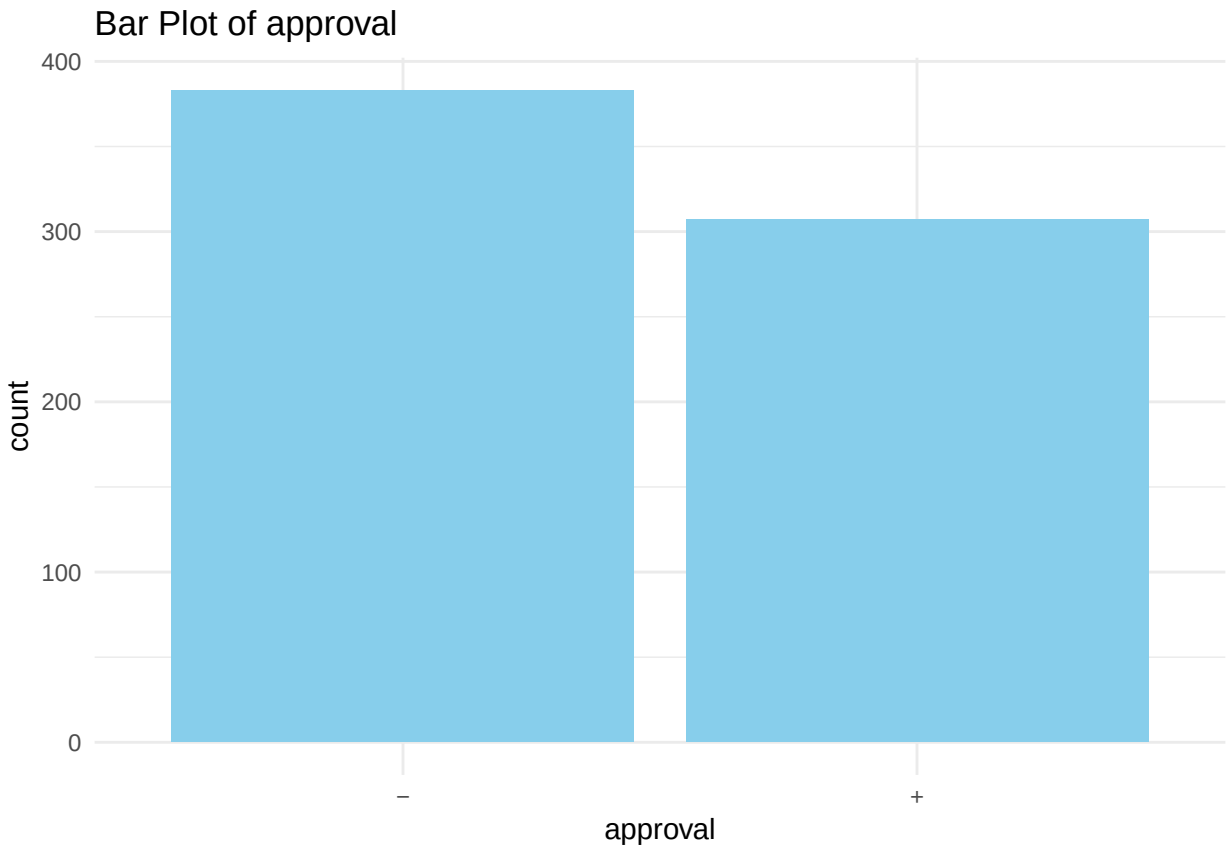

Histogram & Density Plot of cont4
density
0.4
0.3
0.2
0.1
0.0
0
20
40
60
cont4

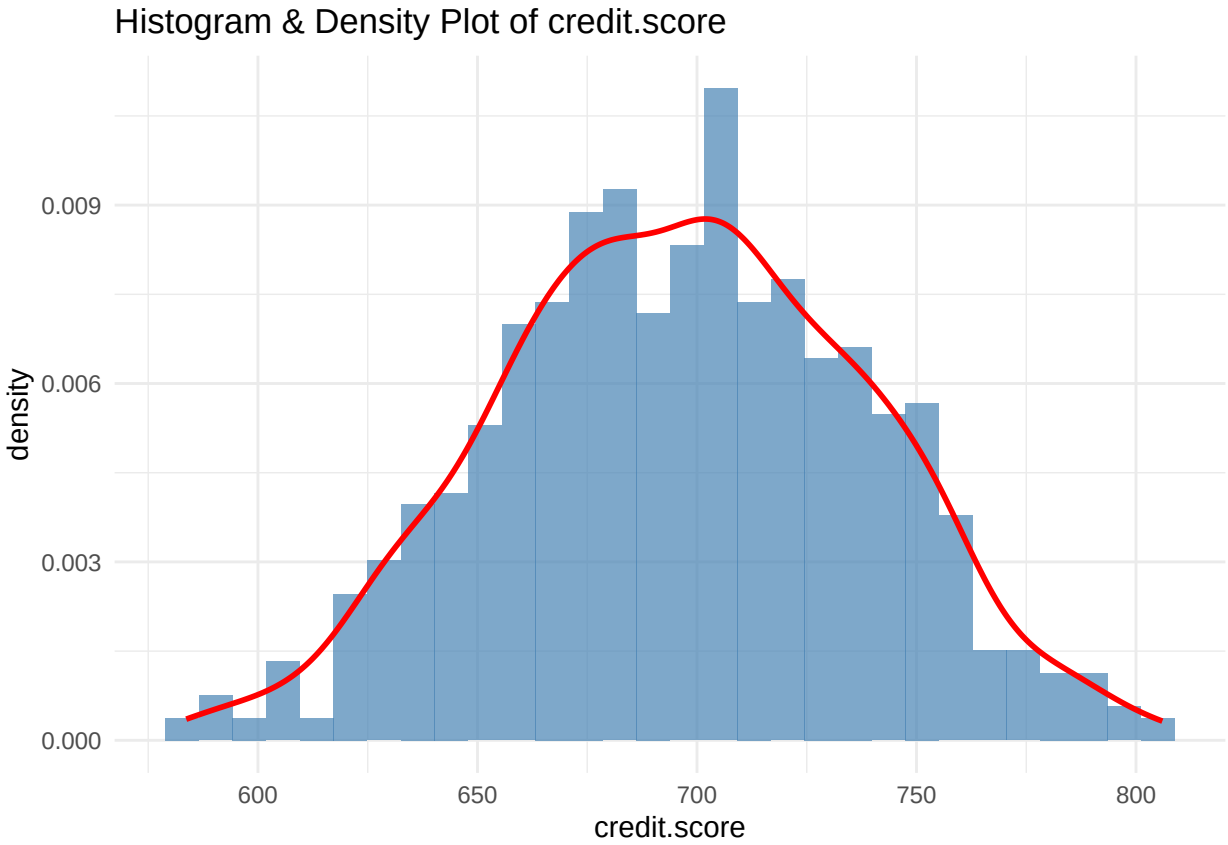


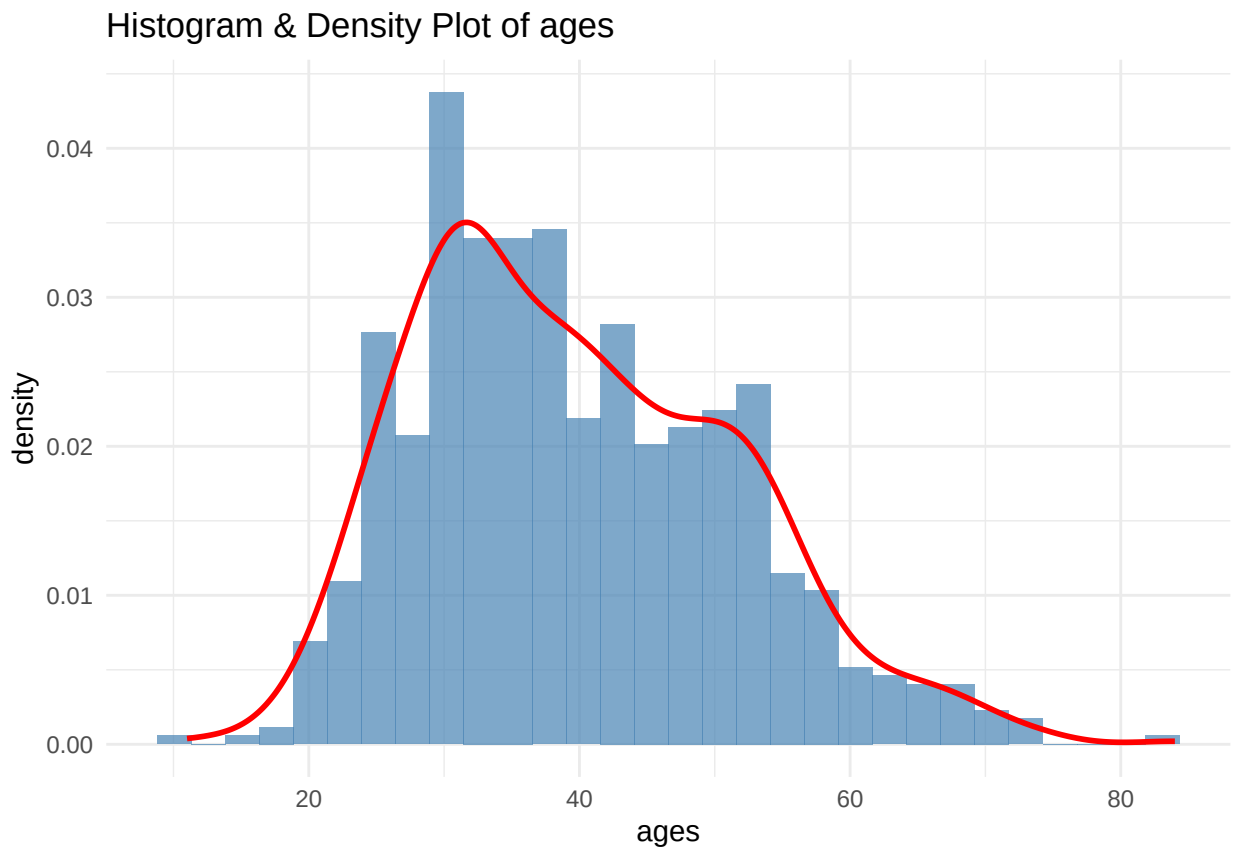

```
## Warning: Removed 13 rows containing non-finite outside the scale range
## (`stat_bin()`).

## Warning: Removed 13 rows containing non-finite outside the scale range
## (`stat_density()`).
```


Histogram & Density Plot of cont5
0.003
0.002
0.001
0.000
density
0
500
1000
1500
2000
cont5


Histogram & Density Plot of cont6
density
0.0020
0.0015
0.0010
0.0005
0.0000
0
25000
50000
75000
100000
cont6


Bar Plot of approval
400
300
200
100
0
count
−
+
approval


Histogram & Density Plot of credit.score
density
0.009
0.006
0.003
0.000
600
650
700
750
800
credit.score




```r
# Question 1b: Applying Normalization
# Z-score normalization for 'credit.score'
df <- df %>%
  mutate(credit_score_zscore = (credit.score - mean(credit.score, na.rm = TRUE)) / sd(credit.score, na.

# Min-Max normalization for 'ages'
df <- df %>%
  mutate(ages_minmax = (ages - min(ages, na.rm = TRUE)) / (max(ages, na.rm = TRUE) - min(ages, na.rm = '

# Decimal scaling for 'cont1'
max_cont1 <- max(abs(df$cont1), na.rm = TRUE)
scaling_factor <- 10^floor(log10(max_cont1) + 1)  # Finds the highest power of 10
df <- df %>%
  mutate(cont1_decimal_scaled = cont1 / scaling_factor)

# Display the transformed dataset
head(df[, c("credit.score", "credit_score_zscore", "ages", "ages_minmax", "cont1", "cont1_decimal_scale
```

```
## # A tibble: 6 x 6
##   credit.score credit_score_zscore  ages ages_minmax cont1 cont1_decimal_scaled
##          <dbl>               <dbl> <dbl>       <dbl> <dbl>                <dbl>
## 1         665.             -0.758     42       0.425  30.8                0.308
## 2         694.             -0.0598    54       0.589  58.7                0.587
## 3         622.             -1.78      29       0.247  24.5                0.245
## 4         654.             -1.01      58       0.644  27.8                0.278
## 5         670.             -0.623     65       0.740  20.2                0.202
```

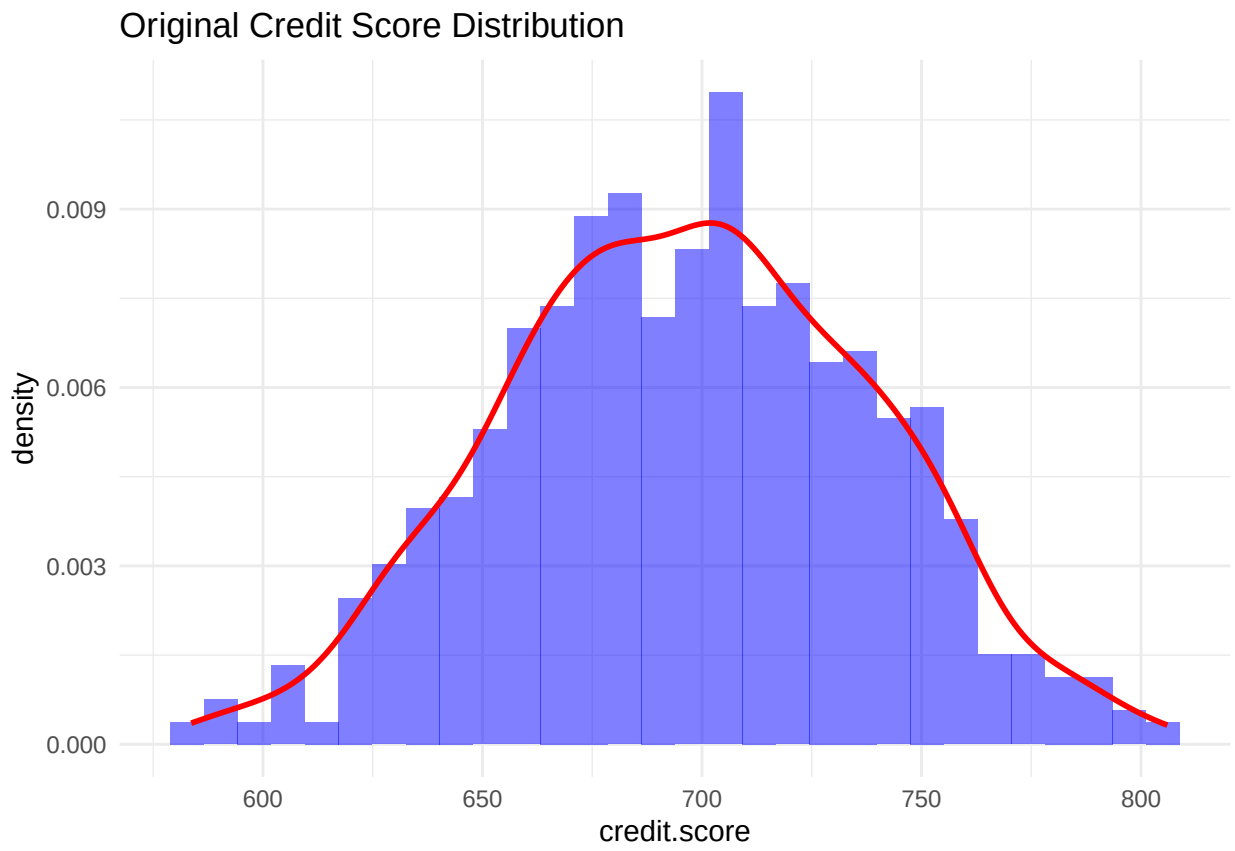
```
## 6         672.             -0.578     61       0.685  32.1                0.321
```

#We applied different normalization techniques based on the distribution and scale of each variable to ensure they are comparable and optimized for machine learning models. Z-score normalization was used for credit.score because it likely follows a normal distribution. This method standardizes values around a mean of zero with a standard deviation of one, making it independent of scale and improving model performance.

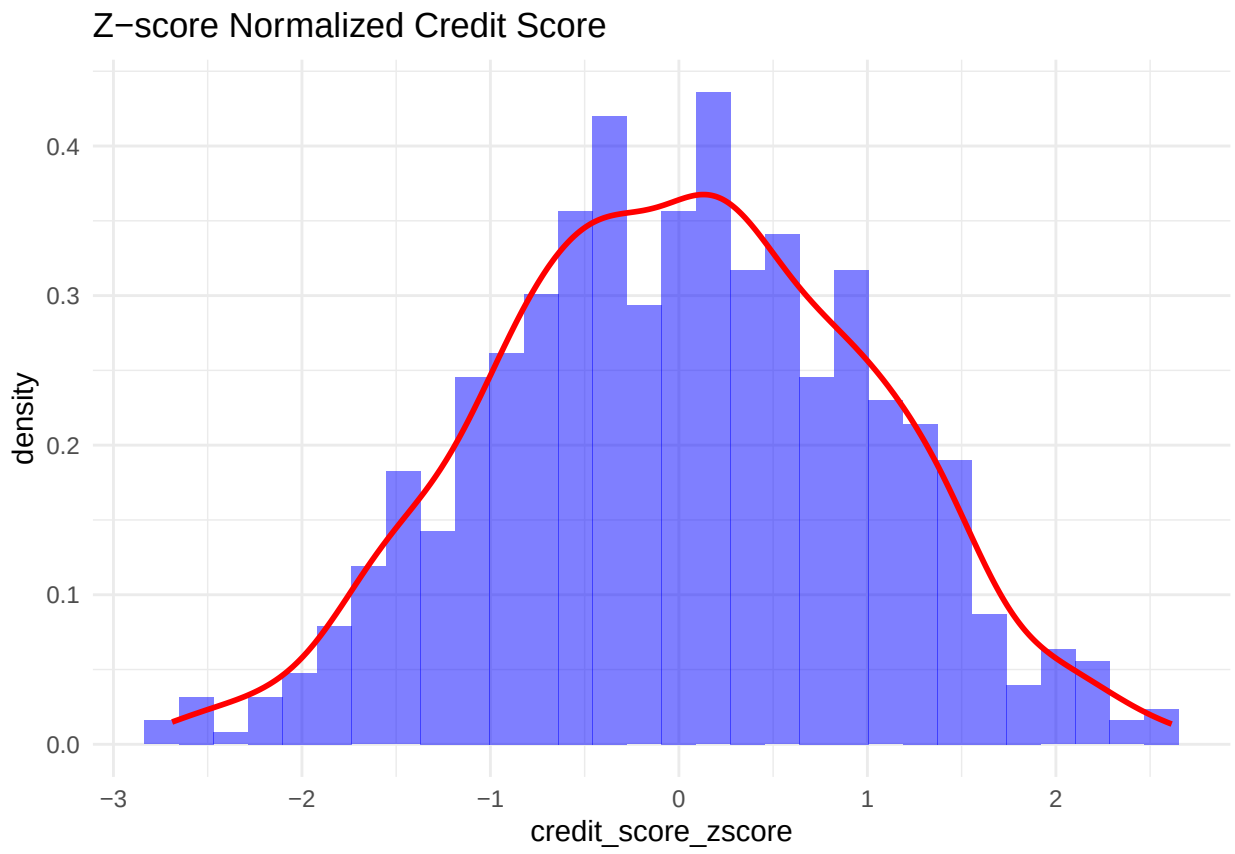
#For ages, we applied Min-Max normalization since it has a fixed range, such as 18 to 80 years. This technique scales values between 0 and 1 while preserving relative differences, ensuring consistency without distorting the original data.

#Lastly, we used decimal scaling for cont1 because it contains large numerical values. By dividing by the highest power of 10, we keep values manageable without altering their distribution. These transformations ensure all features are standardized, making them suitable for models like SVM, which are sensitive to feature magnitudes.

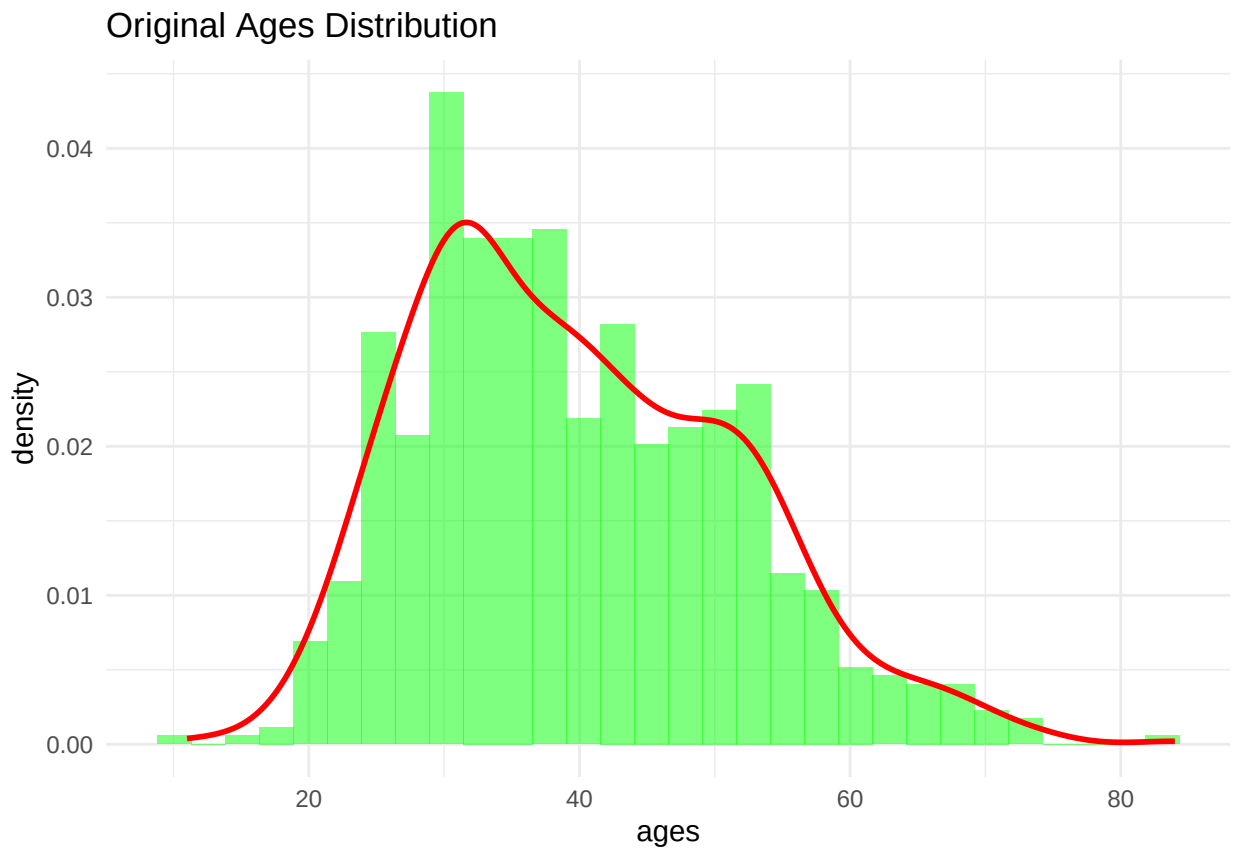
```r
# Question 1c: Visualizing the normalized distributions
p1 <- ggplot(df, aes(x = credit.score)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "blue", alpha = 0.5) +
  geom_density(color = "red", size = 1) +
  ggtitle("Original Credit Score Distribution") +
  theme_minimal()

p2 <- ggplot(df, aes(x = credit_score_zscore)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "blue", alpha = 0.5) +
  geom_density(color = "red", size = 1) +
  ggtitle("Z-score Normalized Credit Score") +
  theme_minimal()

p3 <- ggplot(df, aes(x = ages)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "green", alpha = 0.5) +
  geom_density(color = "red", size = 1) +
  ggtitle("Original Ages Distribution") +
  theme_minimal()

p4 <- ggplot(df, aes(x = ages_minmax)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "green", alpha = 0.5) +
  geom_density(color = "red", size = 1) +
  ggtitle("Min-Max Scaled Ages") +
  theme_minimal()

p5 <- ggplot(df, aes(x = cont1)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "orange", alpha = 0.5) +
  geom_density(color = "red", size = 1) +
  ggtitle("Original Cont1 Distribution") +
  theme_minimal()

p6 <- ggplot(df, aes(x = cont1_decimal_scaled)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "orange", alpha = 0.5) +
  geom_density(color = "red", size = 1) +
  ggtitle("Decimal Scaled Cont1") +
  theme_minimal()

# Print all plots
print(p1)
```


Original Credit Score Distribution
density
0.009
0.006
0.003
0.000
600
650
700
750
800
credit.score

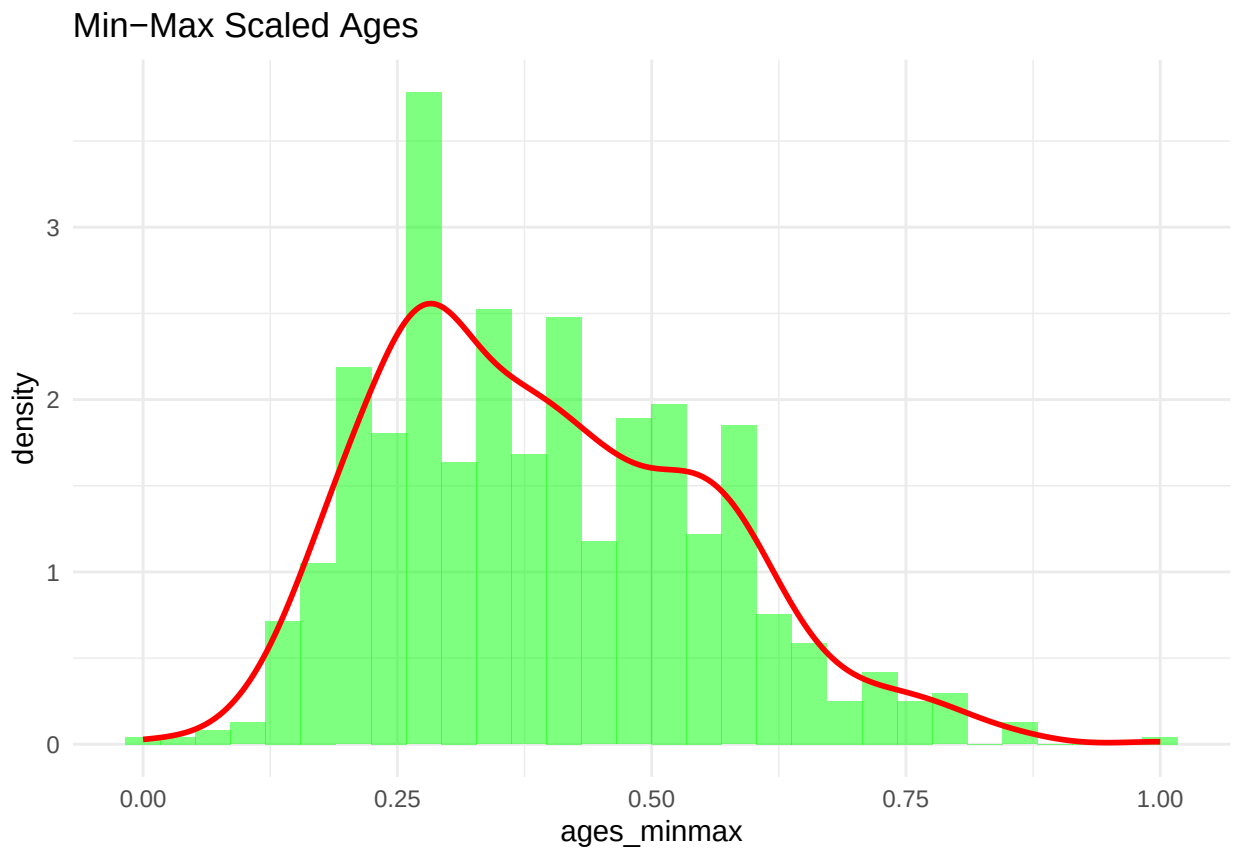
```
print(p2)
```


Z−score Normalized Credit Score
density
0.0
0.1
0.2
0.3
0.4
−3
−2
−1
0
1
2
credit_score_zscore

```
print(p3)
```


Original Ages Distribution
density
0.04
0.03
0.02
0.01
0.00
20
40
60
80
ages

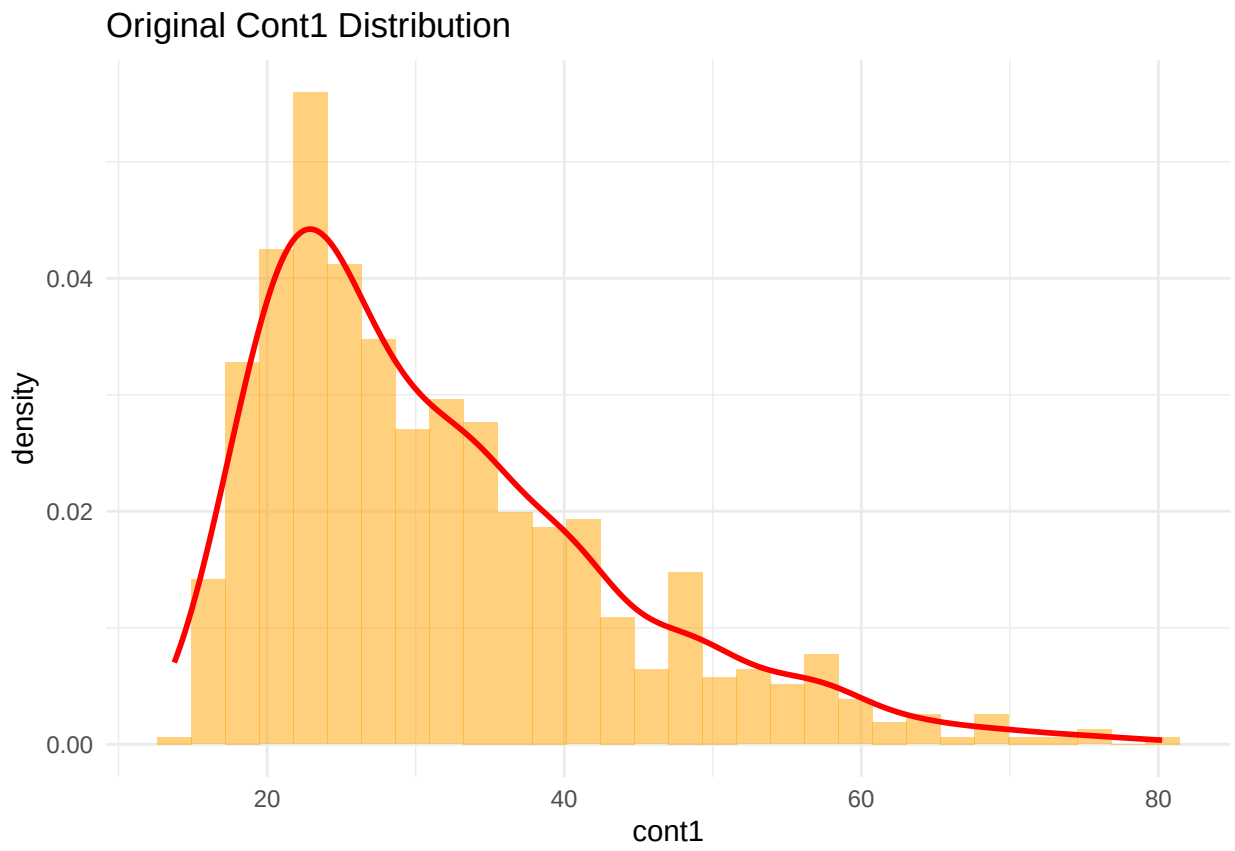
```
print(p4)
```




```
print(p5)
```

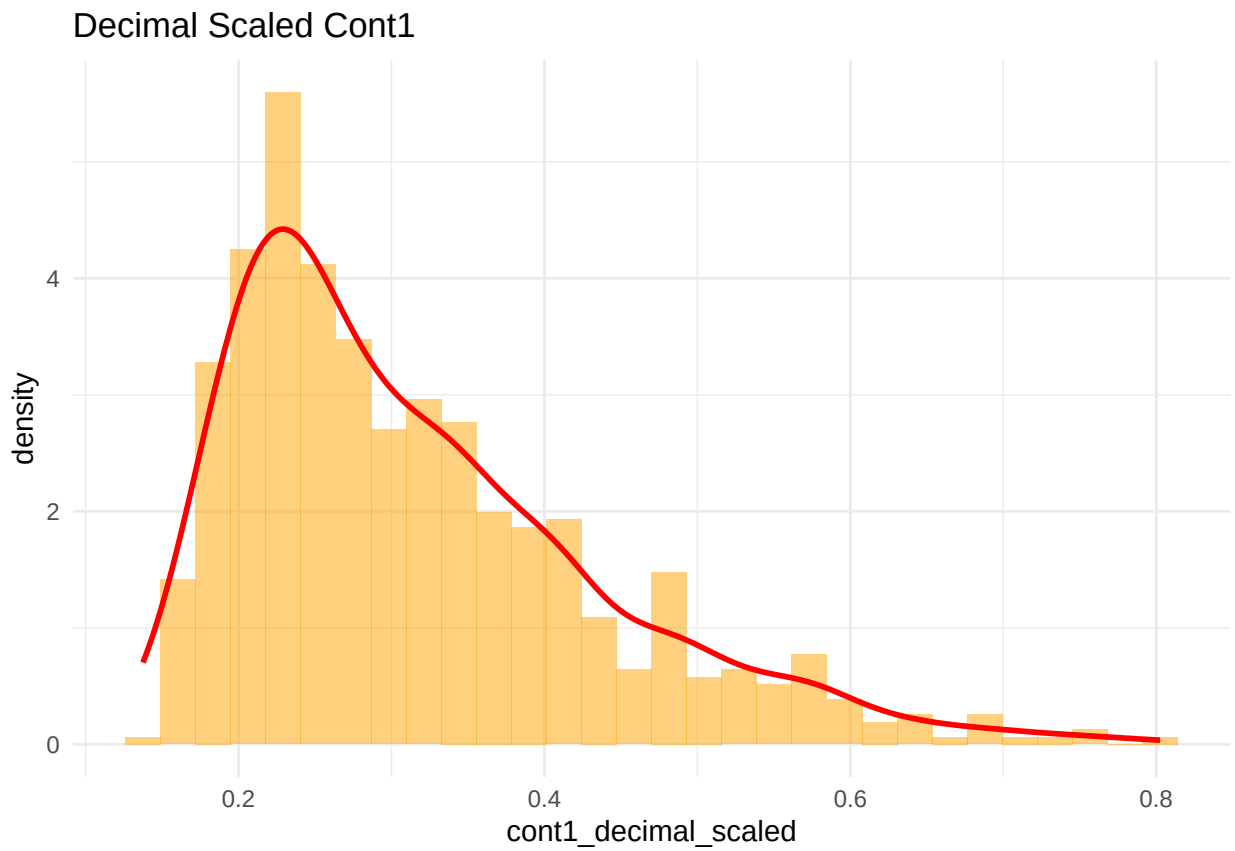
```
## Warning: Removed 12 rows containing non-finite outside the scale range
## (‘stat_bin()‘).
```

```
## Warning: Removed 12 rows containing non-finite outside the scale range
## (‘stat_density()‘).
```




```
print(p6)
```

```
## Warning: Removed 12 rows containing non-finite outside the scale range (‘stat_bin()‘).
## Removed 12 rows containing non-finite outside the scale range
## (‘stat_density()‘).
```



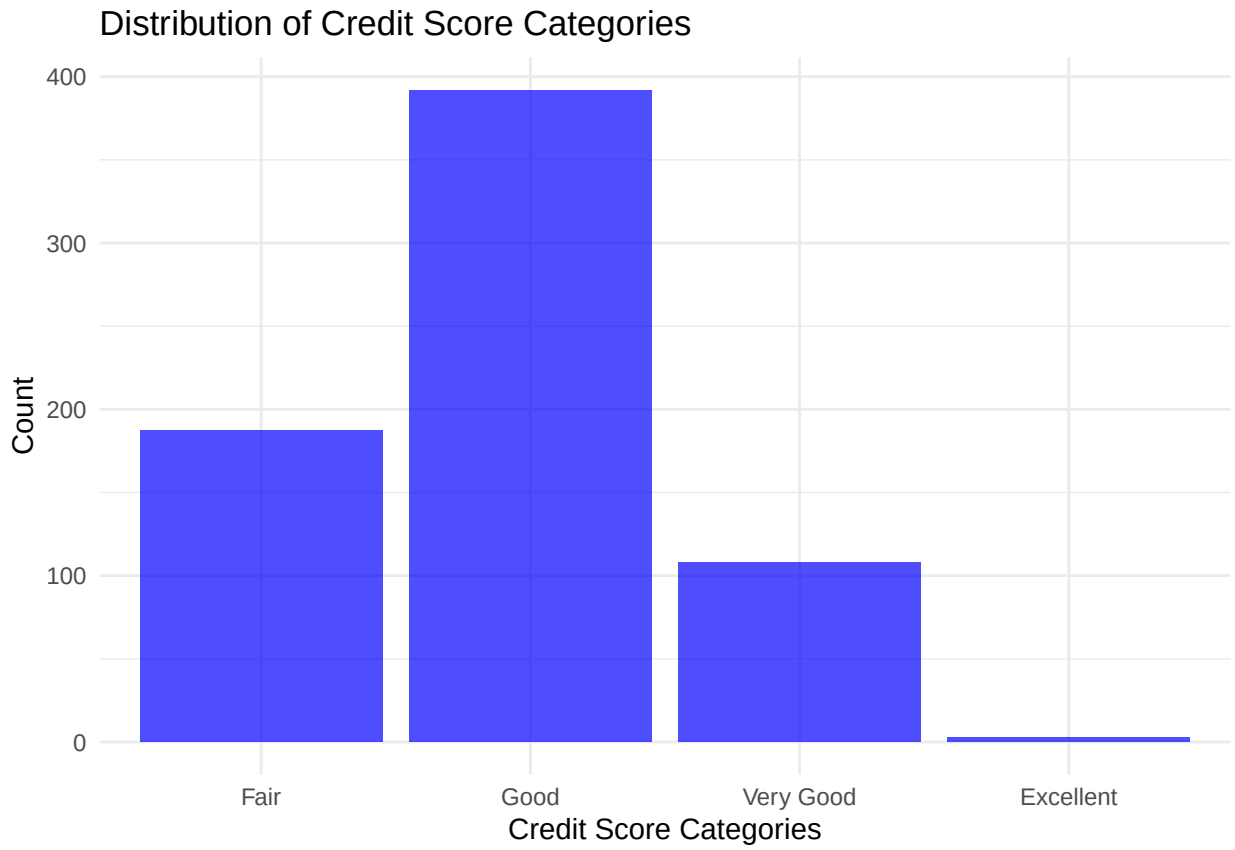
#After normalization, the visualizations show key changes. Z-score normalization (credit.score) centers the mean around 0 with a standard deviation of 1, keeping the shape unchanged. Min-Max normalization (ages) scales values between 0 and 1, compressing the range while maintaining distribution. Decimal scaling (cont1) reduces large values without altering relationships. Overall, normalization ensures a balanced scale for better model performance.

```
#1d
#We categorize credit.score into bins like Poor, Fair, Good, Very Good, and Excellent based on standard
# Binning credit score into risk categories
df <- df %>%
  mutate(credit_bins = case_when(
    credit.score < 580 ~ "Poor",
    credit.score >= 580 & credit.score < 670 ~ "Fair",
    credit.score >= 670 & credit.score < 740 ~ "Good",
    credit.score >= 740 & credit.score < 800 ~ "Very Good",
    credit.score >= 800 ~ "Excellent"
  ))

# Convert the new categorical variable to a factor
df$credit_bins <- factor(df$credit_bins, levels = c("Poor", "Fair", "Good", "Very Good", "Excellent"))

# View the first few rows
head(df[, c("credit.score", "credit_bins")])
```

```
## # A tibble: 6 x 2
##   credit.score credit_bins
##          <dbl> <fct>
```

```
## 1         665. Fair
## 2         694. Good
## 3         622. Fair
## 4         654. Fair
## 5         670. Good
## 6         672. Good
```

```r
# Plot distribution of binned credit scores
ggplot(df, aes(x = credit_bins)) +
  geom_bar(fill = "blue", alpha = 0.7) +
  ggtitle("Distribution of Credit Score Categories") +
  xlab("Credit Score Categories") +
  ylab("Count") +
  theme_minimal()
```



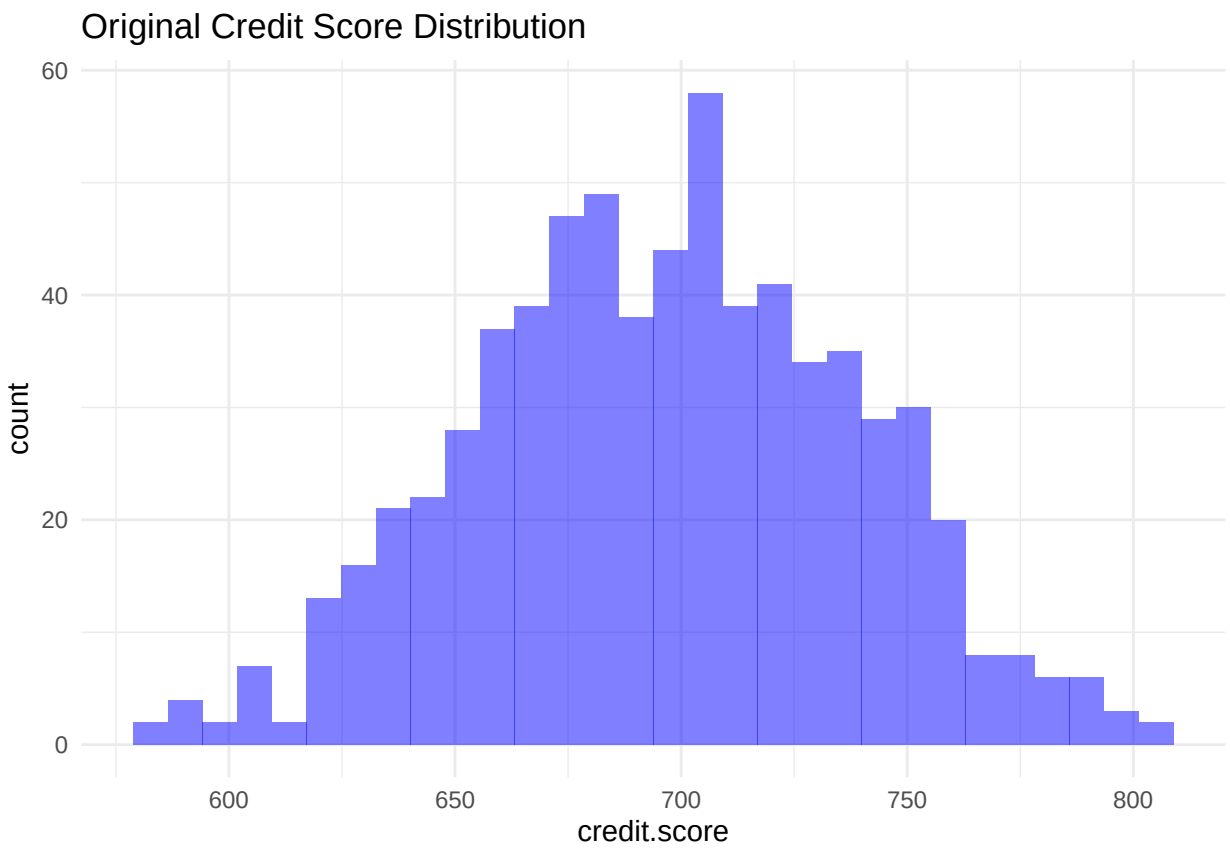

```r
#We chose five bins for credit.score based on industry-standard credit risk categories used by financia
```

```r
#1e
# Define midpoints for each credit score category
df <- df %>%
  mutate(credit_smoothed = case_when(
    credit_bins == "Poor" ~ (300 + 579) / 2,        # Midpoint of 300-579
    credit_bins == "Fair" ~ (580 + 669) / 2,        # Midpoint of 580-669
    credit_bins == "Good" ~ (670 + 739) / 2,        # Midpoint of 670-739
    credit_bins == "Very Good" ~ (740 + 799) / 2,   # Midpoint of 740-799
```

```
    credit_bins == "Excellent" ~ (800 + 850) / 2    # Midpoint of 800-850
  ))

# View the first few rows
head(df[, c("credit.score", "credit_bins", "credit_smoothed")])
```

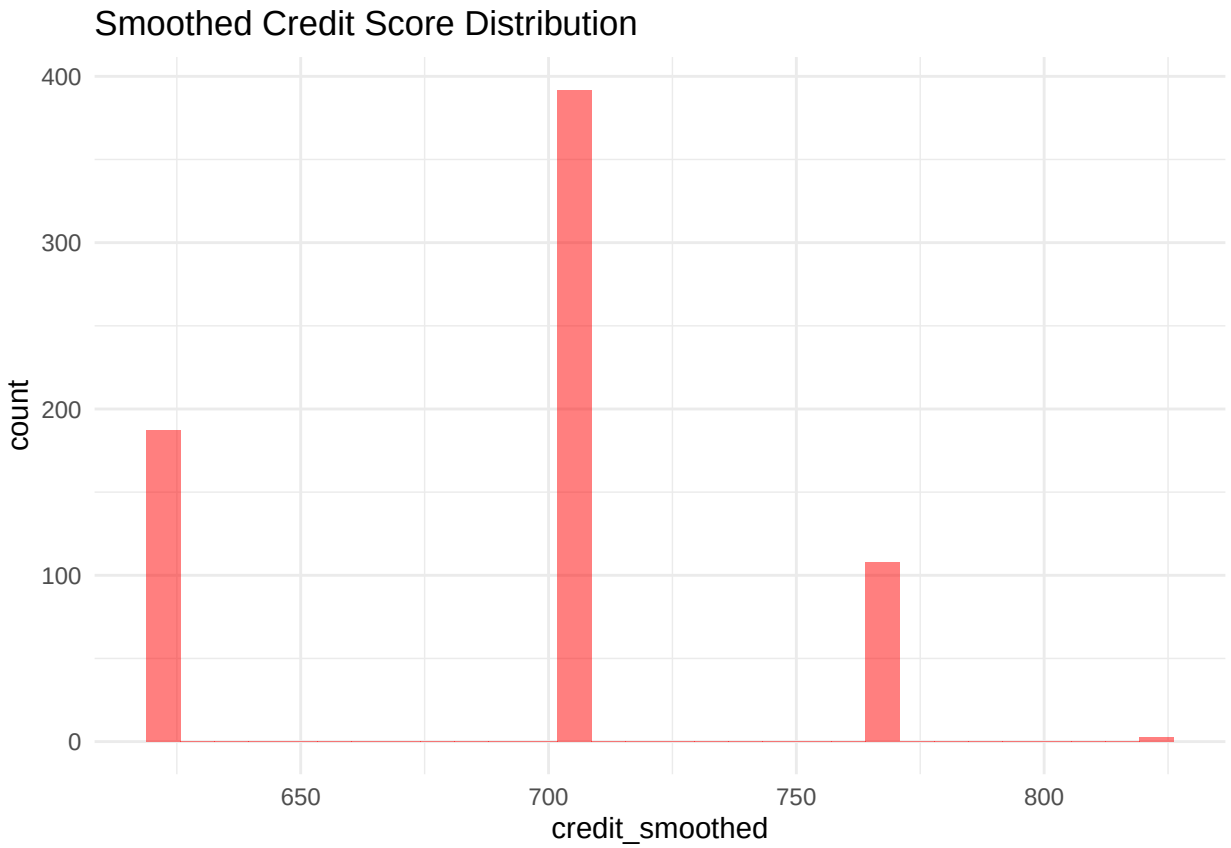
```
## # A tibble: 6 x 3
##   credit.score credit_bins credit_smoothed
##          <dbl> <fct>                <dbl>
## 1         665. Fair                  624.
## 2         694. Good                  704.
## 3         622. Fair                  624.
## 4         654. Fair                  624.
## 5         670. Good                  704.
## 6         672. Good                  704.
```

```
# Histogram of original vs smoothed credit score
ggplot(df, aes(x = credit.score)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.5) +
  ggtitle("Original Credit Score Distribution") +
  theme_minimal()
```




```
ggplot(df, aes(x = credit_smoothed)) +
  geom_histogram(bins = 30, fill = "red", alpha = 0.5) +
```

```
  ggtitle("Smoothed Credit Score Distribution") +
  theme_minimal()
```




```
#We chose midpoint smoothing because it provides a simple, interpretable, and consistent way to convert
```

#2a

```
# Load necessary libraries
library(e1071)   # For SVM
```

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```
library(caret)   # For model training & cross-validation
library(dplyr)   # For data manipulation

# Remove rows with missing values to avoid errors
df <- na.omit(df)

# Convert categorical variables to factors
df$approval <- as.factor(df$approval)
df$bool1 <- as.factor(df$bool1)
df$bool2 <- as.factor(df$bool2)
df$bool3 <- as.factor(df$bool3)
```

```r
# Standardize numerical columns for better SVM performance
num_cols <- sapply(df, is.numeric)
df[num_cols] <- scale(df[num_cols])

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

# Train SVM model with default parameters (C = 1)
svm_model <- train(
  approval ~ .,  # Predict 'approval' using all other features
  data = df,
  method = "svmRadial",  # Radial kernel SVM
  trControl = train_control,
  preProcess = c("center", "scale")  # Standardize data
)

# Print model accuracy and details
print(svm_model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 666 samples
##  16 predictor
##   2 classes: '-', '+'
##
## Pre-processing: centered (19), scaled (19)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 600, 599, 599, 599, 599, 599, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.8604203  0.7229268
##   0.50  0.8649438  0.7315614
##   1.00  0.8575037  0.7160968
##
## Tuning parameter 'sigma' was held constant at a value of 0.04731145
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.04731145 and C = 0.5.
```

```r
# Question 2b: Perform grid search to optimize C parameter
set.seed(123)
tune_grid <- expand.grid(C = seq(0.1, 2, by = 0.1))
svm_model_tuned <- train(approval ~ ., data = df, method = "svmLinear",
                         trControl = trainControl(method = "cv", number = 10),
                         tuneGrid = tune_grid)
print(svm_model_tuned)
```

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

```
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 599, 599, 599, 600, 599, 600, ...
## Resampling results across tuning parameters:
##
##   C    Accuracy   Kappa
##   0.1  0.8633424  0.7286267
##   0.2  0.8633424  0.7286267
##   0.3  0.8633424  0.7286267
##   0.4  0.8633424  0.7286267
##   0.5  0.8633424  0.7286267
##   0.6  0.8633424  0.7286267
##   0.7  0.8633424  0.7286267
##   0.8  0.8633424  0.7286267
##   0.9  0.8633424  0.7286267
##   1.0  0.8633424  0.7286267
##   1.1  0.8633424  0.7286267
##   1.2  0.8633424  0.7286267
##   1.3  0.8633424  0.7286267
##   1.4  0.8633424  0.7286267
##   1.5  0.8633424  0.7286267
##   1.6  0.8633424  0.7286267
##   1.7  0.8633424  0.7286267
##   1.8  0.8633424  0.7286267
##   1.9  0.8633424  0.7286267
##   2.0  0.8633424  0.7286267
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.1.
```

```
#the best parameter chosen through grid search was C = 0.1 using a Linear Kernel.

#The highest accuracy achieved with this parameter was 86.33%, with a Kappa score of 0.7286.

#Interestingly, the accuracy remained constant across all values of C (0.1 to 2.0), suggesting that C h
```

#2c-The accuracy for C = 1 may differ in (b) even though it was included in (a) due to variations in cross-validation splits, as different training and testing partitions can lead to slight performance changes. Additionally, hyperparameter tuning in grid search evaluates multiple values of C, which can influence model optimization differently than using a fixed C = 1. Optimization differences in the SVM solver, along with potential class imbalances in the dataset, may also contribute to these variations. Lastly, some degree of randomness in training, even when using the same parameter, can lead to minor fluctuations in accuracy.

```
# Question 3a: Load and prepare the Star Wars dataset
data(starwars, package = "dplyr")
starwars <- starwars %>% select(-films, -vehicles, -starships, -name) %>% na.omit()

# Convert categorical variables to dummy variables (excluding 'gender')
starwars_dummy <- starwars %>% select(-gender) %>% mutate_if(is.character, as.factor)
starwars_dummy <- model.matrix(~ . -1, data = starwars_dummy)
starwars_final <- data.frame(starwars_dummy, gender = starwars$gender)
```

```r
# Question 3b: Apply SVM to predict gender
set.seed(123)
svm_gender <- train(gender ~ ., data = starwars_final, method = "svmLinear",
                    trControl = trainControl(method = "cv", number = 10))
```

```
## 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
print(svm_gender)
```

```
## Support Vector Machines with Linear Kernel
##
## 29 samples
## 61 predictors
##  2 classes: 'feminine', 'masculine'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 26, 25, 26, 26, 26, 27, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8833333  0.6428571
##
## Tuning parameter 'C' was held constant at a value of 1
```

```r
# Question 3c: Apply PCA for dimensionality reduction
pca_model <- prcomp(starwars_final %>% select(-gender), center = TRUE, scale. = TRUE)
summary(pca_model)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.3651 2.20050 2.11958 2.03331 1.98453 1.94343 1.81337
## Proportion of Variance 0.0917 0.07938 0.07365 0.06778 0.06456 0.06192 0.05391
## Cumulative Proportion  0.0917 0.17108 0.24473 0.31250 0.37707 0.43898 0.49289
##                            PC8     PC9    PC10    PC11   PC12    PC13    PC14
## Standard deviation     1.79736 1.77357 1.72341 1.72051 1.5286 1.47232 1.45253
## Proportion of Variance 0.05296 0.05157 0.04869 0.04853 0.0383 0.03554 0.03459
## Cumulative Proportion  0.54585 0.59741 0.64611 0.69463 0.7329 0.76847 0.80306
##                           PC15    PC16    PC17    PC18    PC19    PC20    PC21
```

```
## Standard deviation     1.35332 1.32507 1.20792 1.16135 1.13040 1.11115 1.04247
## Proportion of Variance 0.03002 0.02878 0.02392 0.02211 0.02095 0.02024 0.01782
## Cumulative Proportion  0.83308 0.86187 0.88579 0.90790 0.92885 0.94909 0.96690
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     1.03195 0.58906 0.52835 0.44929 0.33907 0.10126 0.02936
## Proportion of Variance 0.01746 0.00569 0.00458 0.00331 0.00188 0.00017 0.00001
## Cumulative Proportion  0.98436 0.99005 0.99462 0.99793 0.99982 0.99999 1.00000
##                           PC29
## Standard deviation     7.885e-16
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

```
# Select number of components based on variance explained
starwars_pca <- data.frame(pca_model$x[, 1:5], gender = starwars_final$gender)
```

```
# Question 3d: Train SVM on PCA-transformed data
set.seed(123)
svm_pca <- train(gender ~ ., data = starwars_pca, method = "svmLinear",
                 trControl = trainControl(method = "cv", number = 10))
```

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

```
print(svm_pca)
```

```
## Support Vector Machines with Linear Kernel
##
## 29 samples
##  5 predictor
##  2 classes: 'feminine', 'masculine'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 26, 25, 26, 26, 26, 27, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7916667  0.2142857
##
## Tuning parameter 'C' was held constant at a value of 1
```

#Bonus Questions

```
# Question 4a: Load and Explore Sacramento Housing Data
data(Sacramento, package = "caret")

# Remove 'zip' and 'city' columns
Sacramento <- Sacramento %>% select(-zip, -city)

# Check class balance
table(Sacramento$type)
```

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

```
# Question 4b: Normalize 'price' using log transformation
Sacramento <- Sacramento %>%
  mutate(log_price = log(price))
```

```
# Question 4c: Apply SVM to predict 'type' and evaluate with grid search
set.seed(123)
tune_grid <- expand.grid(C = seq(0.1, 2, by = 0.1))
train_control <- trainControl(method = "cv", number = 10)
svm_model <- train(type ~ ., data = Sacramento, method = "svmLinear",
                   trControl = train_control, tuneGrid = tune_grid)
print(svm_model)
```

```
## Support Vector Machines with Linear Kernel
##
## 932 samples
##   7 predictor
##   3 classes: 'Condo', 'Multi_Family', 'Residential'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 838, 838, 839, 839, 839, 838, ...
## Resampling results across tuning parameters:
##
##   C    Accuracy   Kappa
##   0.1  0.9292024  0.00000000
##   0.2  0.9292024  0.00000000
##   0.3  0.9302776  0.02366621
##   0.4  0.9302776  0.02366621
##   0.5  0.9302776  0.04565832
##   0.6  0.9313529  0.06716543
##   0.7  0.9302776  0.06332114
##   0.8  0.9302776  0.10598716
##   0.9  0.9313529  0.13333091
##   1.0  0.9324167  0.15701156
##   1.1  0.9334805  0.17601384
##   1.2  0.9334805  0.17601384
##   1.3  0.9334805  0.17601384
##   1.4  0.9334805  0.17601384
##   1.5  0.9324053  0.17278342
##   1.6  0.9324053  0.17278342
##   1.7  0.9324053  0.17278342
##   1.8  0.9324053  0.17278342
##   1.9  0.9324053  0.17278342
##   2.0  0.9324053  0.17278342
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.1.
```

```r
# Question 5: Partition 'mtcars' into 5 folds and visualize 'gears' distribution
set.seed(123)
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(x = factor(folds), fill = factor(gear))) +
  geom_bar(position = "dodge") +
  ggtitle("Distribution of 'gears' Across 5 Folds") +
  theme_minimal()
```