Homework 2 - Fundamentals of Data Science

Laurentiu Mandocescu

2025-02-05

Problem 1: Data Preprocessing

Part (a): Load and Inspect the Data

```
# Load necessary libraries
library(tidyverse)

# Read the dataset
bank_data <- read.csv("BankData.csv")

# View first few rows
head(bank_data)
```

```
##   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
```

```
# Structure of the data
str(bank_data)
```

```
## 'data.frame':    690 obs. of  13 variables:
##  $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ cont1       : num  30.8 58.7 24.5 27.8 20.2 ...
##  $ cont2       : num  0 4.46 0.5 1.54 5.62 ...
##  $ cont3       : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ bool1       : chr  "t" "t" "t" "t" ...
##  $ bool2       : chr  "t" "t" "f" "t" ...
##  $ cont4       : int  1 6 0 5 0 0 0 0 0 0 ...
```

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

```
# Summary statistics
summary(bank_data)
```

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

```
# Check for missing values
colSums(is.na(bank_data))
```

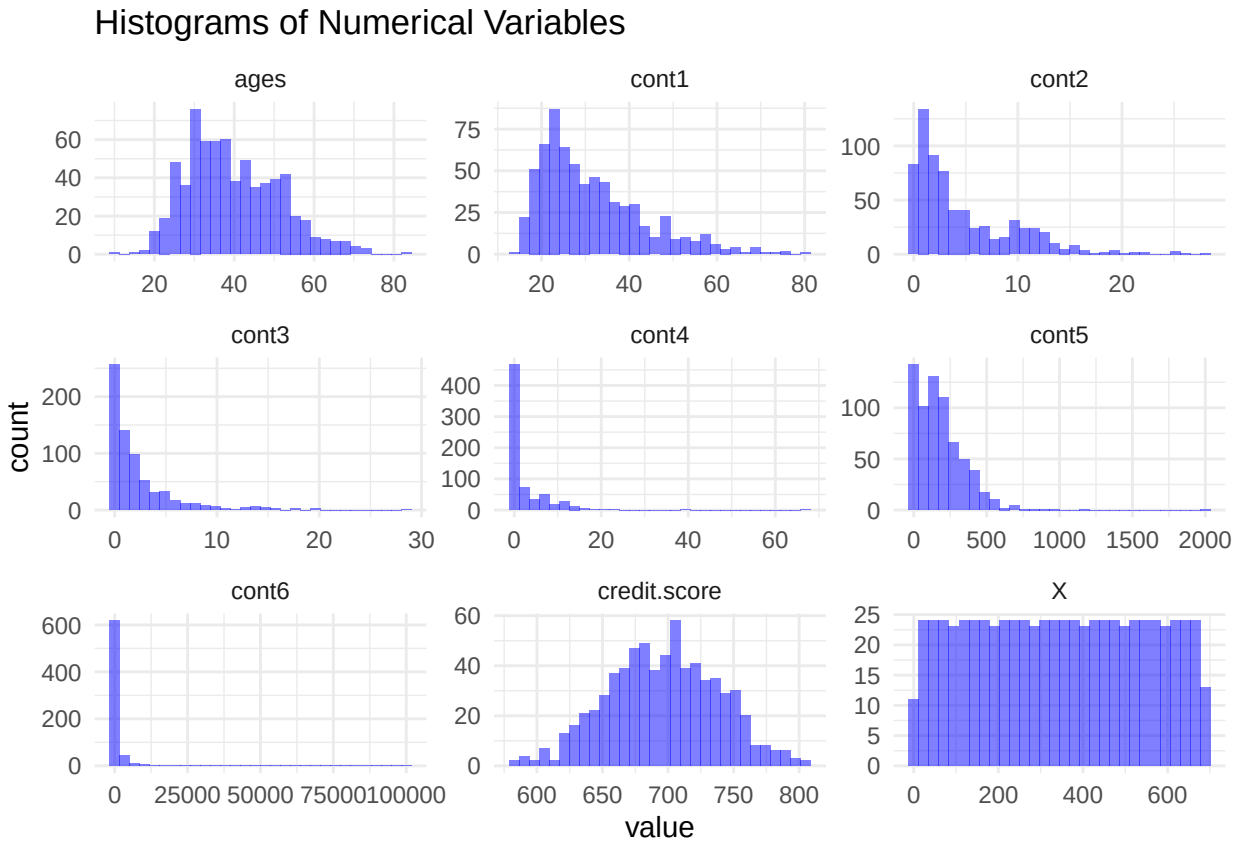
```
##            X        cont1        cont2        cont3        bool1        bool2
##            0           12            0            0            0            0
##        cont4        bool3        cont5        cont6     approval credit.score
##            0            0           13            0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

Part (a): Visualizing Distributions

Numerical Variables

```
# Histograms and Density Plots for Numerical Variables
bank_data %>%
  select(where(is.numeric)) %>%
  gather() %>%
  ggplot(aes(value)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.5) +
  facet_wrap(~key, scales = "free") +
  theme_minimal() +
  ggtitle("Histograms of Numerical Variables")
```

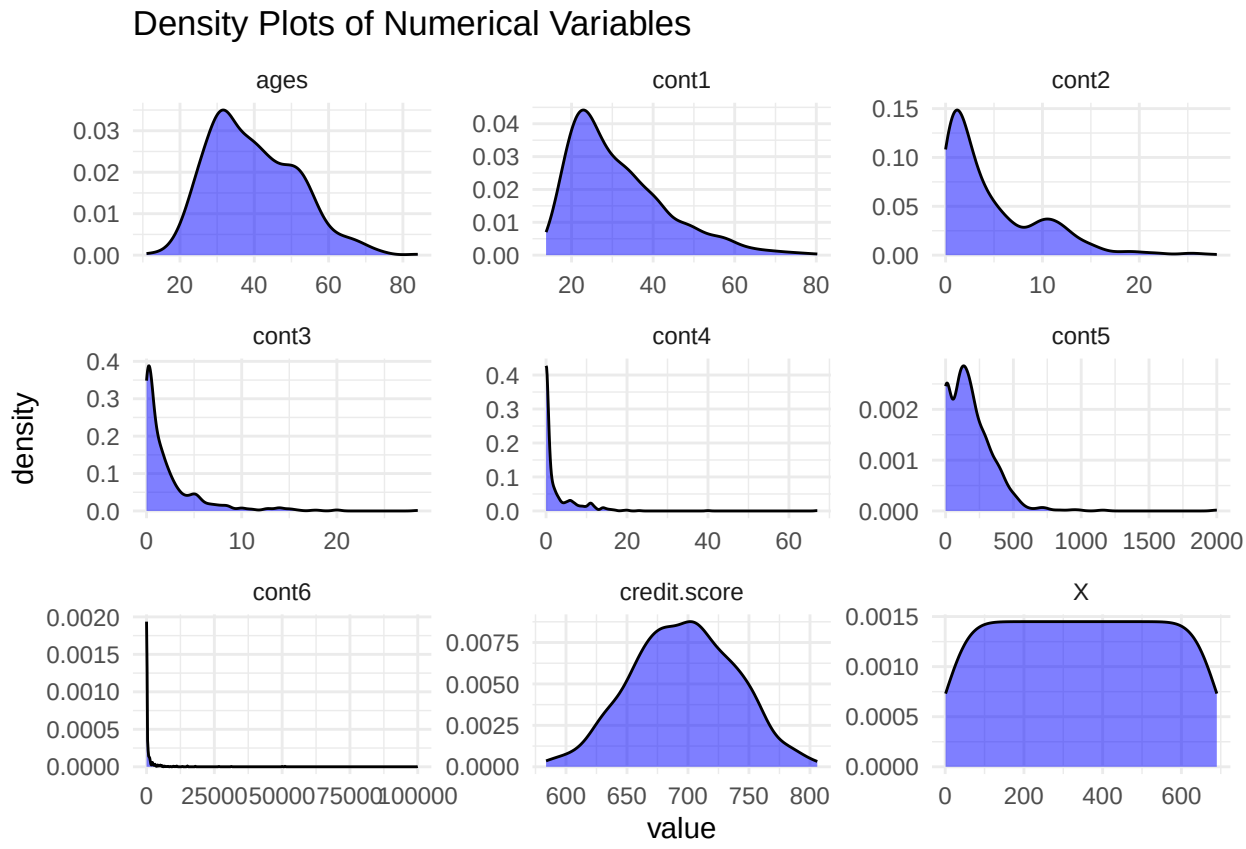
```
## Warning: Removed 25 rows containing non-finite outside the scale range
## (`stat_bin()`).
```




```
# Density Plots
bank_data %>%
  select(where(is.numeric)) %>%
  gather() %>%
  ggplot(aes(value)) +
  geom_density(fill = "blue", alpha = 0.5) +
  facet_wrap(~key, scales = "free") +
```

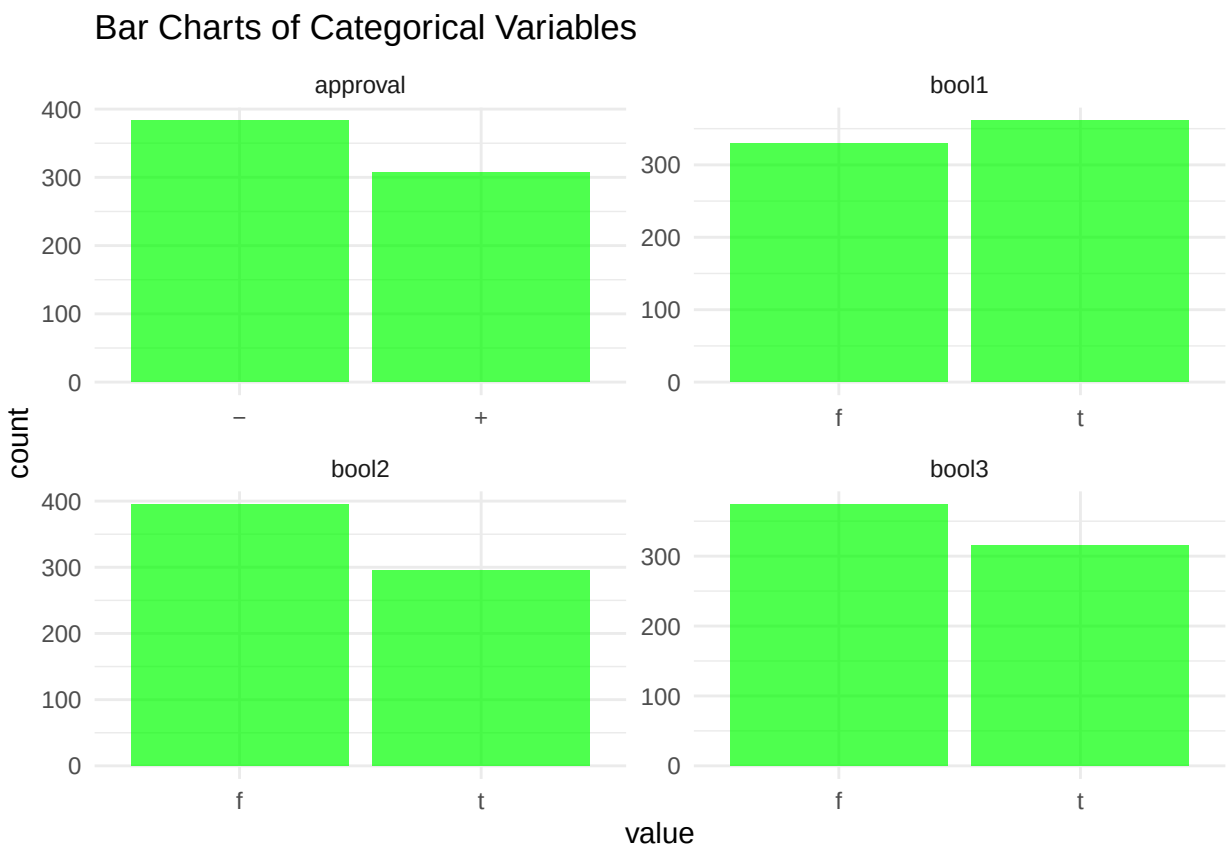
```
  theme_minimal() +
  ggtitle("Density Plots of Numerical Variables")
```

```
## Warning: Removed 25 rows containing non-finite outside the scale range
## (`stat_density()`).
```



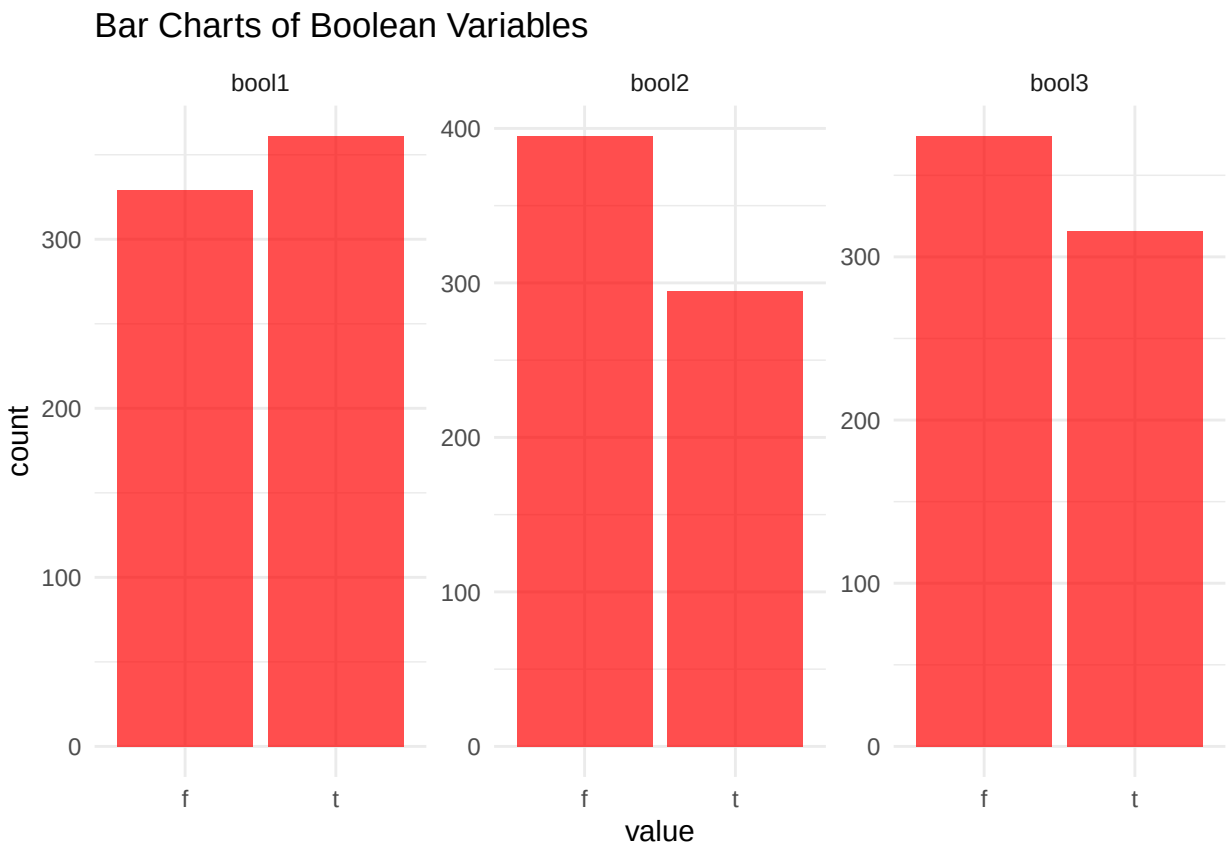
Categorical Variables

```
# Bar Charts for Categorical Variables
bank_data %>%
  select(where(is.character)) %>%
  gather() %>%
  ggplot(aes(value)) +
  geom_bar(fill = "green", alpha = 0.7) +
  facet_wrap(~key, scales = "free") +
  theme_minimal() +
  ggtitle("Bar Charts of Categorical Variables")
```



Boolean Variables

```
# Bar Charts for Boolean Variables
bank_data %>%
  select(starts_with("bool")) %>%
  gather() %>%
  ggplot(aes(value)) +
  geom_bar(fill = "red", alpha = 0.7) +
  facet_wrap(~key, scales = "free") +
  theme_minimal() +
  ggtitle("Bar Charts of Boolean Variables")
```



Part (b): Normalization of Variables

Applying Normalization Techniques

```r
# Z-score Normalization for cont1
bank_data$cont1_z <- scale(bank_data$cont1)

# Min-Max Normalization for credit.score
bank_data$credit.score_minmax <- (bank_data$credit.score - min(bank_data$credit.score)) /
                                 (max(bank_data$credit.score) - min(bank_data$credit.score))

# Decimal Scaling for cont6
factor <- 10^ceiling(log10(max(abs(bank_data$cont6))))
bank_data$cont6_decimal <- bank_data$cont6 / factor

# Display summary of transformed variables
summary(bank_data[, c("cont1_z", "credit.score_minmax", "cont6_decimal")])
```

```
##      cont1_z.V1      credit.score_minmax cont6_decimal
##  Min.   :-1.490080   Min.   :0.0000      Min.   :0.000000
##  1st Qu.:-0.749772   1st Qu.:0.3737      1st Qu.:0.000000
##  Median :-0.259927   Median :0.5110      Median :0.000050
##  Mean   : 0.000000   Mean   :0.5070      Mean   :0.010174
```

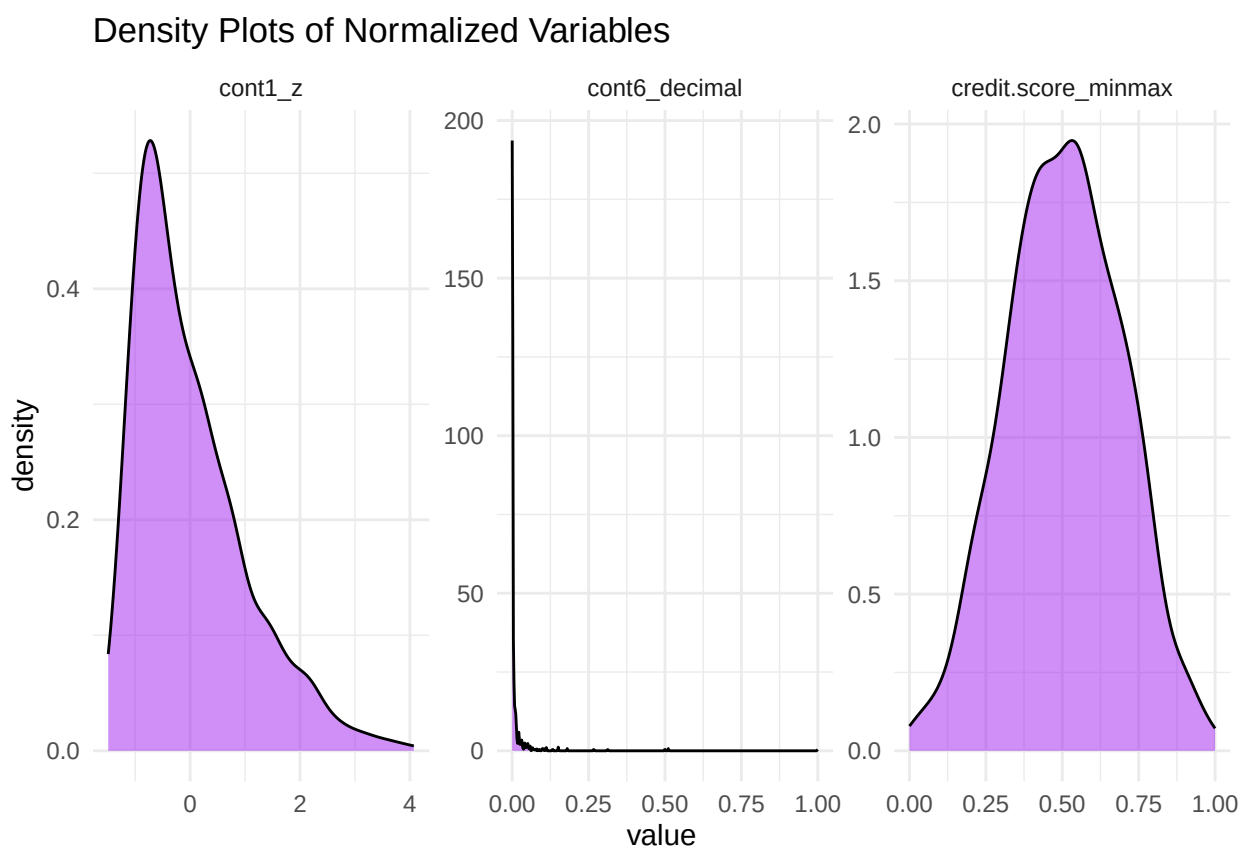
```
##  3rd Qu.: 0.557109   3rd Qu.:0.6418      3rd Qu.:0.003955
##  Max.   : 4.071115   Max.   :1.0000      Max.   :1.000000
##  NA's   :12
```

Visualization of Normalized Variables

```
# Density Plots of Normalized Variables
bank_data %>%
  select(cont1_z, credit.score_minmax, cont6_decimal) %>%
  gather() %>%
  ggplot(aes(value)) +
  geom_density(fill = "purple", alpha = 0.5) +
  facet_wrap(~key, scales = "free") +
  theme_minimal() +
  ggtitle("Density Plots of Normalized Variables")
```

```
## Warning: attributes are not identical across measure variables; they will be
## dropped
```

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



I chose min-max normalization for `credit.score_minmax` because credit scores are usually within a set range. I did Min-max normalization to transform these values into a 0 to 1 scale while keeping their relative positioning. This allows for easier comparisons without distorting the relationships between data points.

I applied Z-score normalization to `cont1_z` because the variable follows a somewhat normal distribution but has some skewness. Standardizing the data ensures that it has a mean of zero and a standard deviation of one, which in turn makes it more suitable for models that will assume normality. This transformation keeps the shape of the original distribution while it expresses each value in terms of standard deviations from the mean.

For `cont6_decimal`, I selected decimal scaling due to its extremely wide range of values. Without normalization, this variable might end up dominating calculations and skew the results. I'm thinking that, by dividing by a power of ten, decimal scaling reduces its influence while maintaining proportional relationships among data points.

Observed Changes in Distribution After Normalization

Z-score normalization adjusted `cont1` by shifting its mean to zero and scaling based on standard deviation. This transformation ensures that values are centered while maintaining the overall shape of the distribution. The primary change observed is the standardization of variance, which helps in ensuring more stable model training.

Min-max normalization transformed `credit.score` into a range of 0 to 1. The distribution remains similar, but all values are now scaled proportionally within this new range. This change ensures that the variable aligns well with other features without altering relative differences between data points.

Decimal scaling compressed the range of `cont6` significantly. Since this variable had a large magnitude, this normalization method effectively reduced its impact in calculations. The distribution still exhibits skewness, but the extreme values no longer dominate the dataset, making it more comparable to other features.

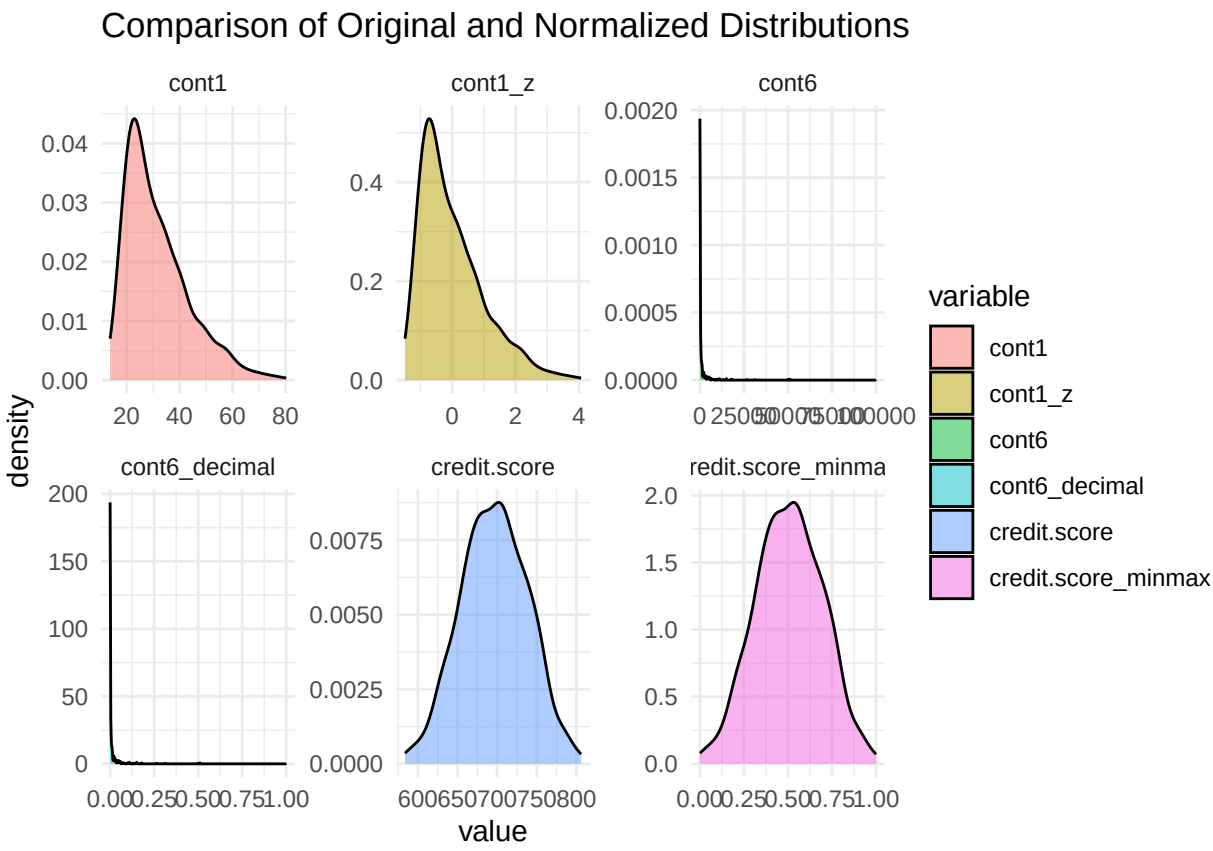
Overall, these normalization techniques ensure that numerical variables are brought to comparable scales, improving the interpretability and usability of the data for subsequent analysis or machine learning applications.

Part (c): Visualization of Normalized Variables

```
# Comparing Original and Normalized Distributions
bank_data %>%
  select(cont1, cont1_z, credit.score, credit.score_minmax, cont6, cont6_decimal) %>%
  gather(key = "variable", value = "value") %>%
  ggplot(aes(value, fill = variable)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~variable, scales = "free") +
  theme_minimal() +
  ggtitle("Comparison of Original and Normalized Distributions")
```

```
## Warning: attributes are not identical across measure variables; they will be
## dropped
```

```
## Warning: Removed 24 rows containing non-finite outside the scale range
## (`stat_density()`).
```



What has changed from the previous visualization?

Z-score Normalization `cont1_z` is centered around 0, and has a standard deviation of 1, compared to `cont1` which had a right skew with a broad range. `cont1_z` is easier to compare with other standardized variables. The shape of `court1_z` remained similar but the difference is in the extreme values which are now adjusted relative to the their deviation from the mean.

Min-max normalizations `credit.score_minmax` is transformed to fit within 0 and 1, which preserves the relative positioning while ensuring consistency with other variables. The shape of the distribution remained the same, but on a standardized scale.

Decimal scaling `cont6_decimal` has the highest value of 1, now after normalization, whereas `cont6` had a large range with extreme values. The distribution still appears skewed. but its impact on the data set is reduced.

Part (d): Binning a Numerical Variable

```
# Check summary statistics for numerical variables
summary(bank_data %>% select(where(is.numeric)))
```

```
##        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
##      cont4          cont5          cont6          credit.score
##  Min.   : 0.0   Min.   :   0   Min.   :     0.0   Min.   :583.7
##  1st Qu.: 0.0   1st Qu.:  75   1st Qu.:     0.0   1st Qu.:666.7
##  Median : 0.0   Median : 160   Median :     5.0   Median :697.3
##  Mean   : 2.4   Mean   : 184   Mean   :  1017.4   Mean   :696.4
##  3rd Qu.: 3.0   3rd Qu.: 276   3rd Qu.:   395.5   3rd Qu.:726.4
##  Max.   :67.0   Max.   :2000   Max.   :100000.0   Max.   :806.0
##                 NA's   :13
##       ages           cont1_z.V1      credit.score_minmax cont6_decimal
##  Min.   :11.00   Min.   :-1.490080   Min.   :0.0000      Min.   :0.000000
##  1st Qu.:31.00   1st Qu.:-0.749772   1st Qu.:0.3737      1st Qu.:0.000000
##  Median :38.00   Median :-0.259927   Median :0.5110      Median :0.000050
##  Mean   :39.67   Mean   : 0.000000   Mean   :0.5070      Mean   :0.010174
##  3rd Qu.:48.00   3rd Qu.: 0.557109   3rd Qu.:0.6418      3rd Qu.:0.003955
##  Max.   :84.00   Max.   : 4.071115   Max.   :1.0000      Max.   :1.000000
##                  NA's   :12
```

After reviewing the summary statistics, I chose `cont1` for binning because it has a well-distributed range of values that allow for structured segmentation.

```
# Implementing Equal-Depth Binning for 'cont1'
num_bins <- 3
bin_breaks <- quantile(bank_data$cont1, probs = seq(0, 1, length.out = num_bins + 1), na.rm = TRUE)
bank_data$cont1_bins <- cut(bank_data$cont1, breaks = bin_breaks, include.lowest = TRUE, labels = c("Lo

# Display first few rows
head(bank_data[, c("cont1", "cont1_bins")])
```

```
##   cont1 cont1_bins
## 1 30.83     Medium
## 2 58.67       High
## 3 24.50     Medium
## 4 27.83     Medium
## 5 20.17        Low
## 6 32.08     Medium
```

I applied equal-depth binning to cont1 to ensure each bin has an approximately equal number of observations. This method helps balance data distribution and avoids sparse bins.

Low: Lower third of values. Medium: Middle third of values. High: Upper third of values.

I chose equal-depth binning because it adapts to the data distribution, ensuring balanced segmentation even in skewed datasets.

Part (e): Smoothing a Numerical Variable

```
# Bin Mean Smoothing (using cont1_bins)
bank_data <- bank_data %>%
```

```
  group_by(cont1_bins) %>%
  mutate(cont1_smooth = mean(cont1, na.rm = TRUE)) %>%
  ungroup()

# Display first few rows
head(bank_data[, c("cont1", "cont1_bins", "cont1_smooth")])
```

```
## # A tibble: 6 x 3
##   cont1 cont1_bins cont1_smooth
##   <dbl> <fct>             <dbl>
## 1  30.8 Medium             28.8
## 2  58.7 High               45.5
## 3  24.5 Medium             28.8
## 4  27.8 Medium             28.8
## 5  20.2 Low                20.5
## 6  32.1 Medium             28.8
```

Bin mean smoothing replaces each value in `cont1` with the mean of its corresponding bin in `cont1_bins`. This reduced the noise while it preserve the structure of the data, as the smoothed values are derived directly from the binned variable. The smoothed version is a numerical representation of the binned variable which is better suited for further analysis.

Problem 2

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

```
# 1. Data preparation
# Remove rows with missing values
bank_data_clean <- na.omit(bank_data)

# Select relevant features for the model
bank_data_model <- bank_data_clean[, c("cont1", "cont2", "cont3", "cont4", "cont5", "cont6", "approval"]

# Convert approval to factor
bank_data_model$approval <- as.factor(bank_data_model$approval)
```

```
# 2. Implement 10-fold cross-validation with SVM
set.seed(123)  # for reproducibility
folds <- 10
cv_indices <- sample(1:folds, nrow(bank_data_model), replace = TRUE)
cv_accuracy <- numeric(folds)

# Perform cross-validation
for(i in 1:folds) {
    test_indices <- which(cv_indices == i)
    train_data <- bank_data_model[-test_indices, ]
    test_data <- bank_data_model[test_indices, ]

    # Train SVM with default C=1
    svm_model <- svm(approval ~ ., data = train_data,
```

```
                    kernel = "linear", cost = 1)

    # Predict and calculate accuracy
    predictions <- predict(svm_model, test_data)
    cv_accuracy[i] <- mean(predictions == test_data$approval)
}

# Report results
cat("SVM with 10-fold CV (C=1):\n")
```

```
## SVM with 10-fold CV (C=1):
```

```
cat("Mean Accuracy:", mean(cv_accuracy), "\n")
```

```
## Mean Accuracy: 0.7581588
```

```
cat("Standard Deviation:", sd(cv_accuracy), "\n")
```

```
## Standard Deviation: 0.06807005
```

Part 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?

```
# Define a range of C values to test
# Using a logarithmic scale is common practice for SVM's C parameter
c_values <- c(0.001, 0.01, 0.1, 1, 10, 100)

# Store results for each C value
grid_results <- matrix(0, nrow = length(c_values), ncol = 2)
colnames(grid_results) <- c("C", "Accuracy")

# Perform grid search with cross-validation
set.seed(123)  # maintain reproducibility
for(j in 1:length(c_values)) {
    cv_accuracy <- numeric(folds)

    for(i in 1:folds) {
        test_indices <- which(cv_indices == i)
        train_data <- bank_data_model[-test_indices, ]
        test_data <- bank_data_model[test_indices, ]

        svm_model <- svm(approval ~ ., data = train_data,
                         kernel = "linear", cost = c_values[j])

        predictions <- predict(svm_model, test_data)
        cv_accuracy[i] <- mean(predictions == test_data$approval)
    }

    grid_results[j,] <- c(c_values[j], mean(cv_accuracy))
}
```

```r
# Find optimal C value
best_row <- which.max(grid_results[,"Accuracy"])
cat("Grid Search Results:\n")
```

```
## Grid Search Results:
```

```r
print(grid_results)
```

```
##          C  Accuracy
## [1,] 1e-03 0.5919223
## [2,] 1e-02 0.7395921
## [3,] 1e-01 0.7598096
## [4,] 1e+00 0.7581588
## [5,] 1e+01 0.7592120
## [6,] 1e+02 0.7592120
```

```r
cat("\nOptimal C:", grid_results[best_row, "C"],
    "\nBest Accuracy:", grid_results[best_row, "Accuracy"])
```

```
##
## Optimal C: 0.1
## Best Accuracy: 0.7598096
```

The grid search over C values {0.001, 0.01, 0.1, 1, 10, 100} show:

Optimal C parameter: 0.1 Best accuracy achieved: 75.98%

Looking at the accuracy pattern: Very small C (0.001): Poor performance (59.19%) C = 0.1: Best performance (75.98%) Larger C values (10, 100): Slightly worse but stable (75.92%)

This shows that C = 0.1 provides the best balance between model flexibility and generalization.

```r
# Try different seeds for C=1
set.seed(456)  # New seed
cv_indices_new <- sample(1:folds, nrow(bank_data_model), replace = TRUE)
cv_accuracy_new <- numeric(folds)

for(i in 1:folds) {
    test_indices <- which(cv_indices_new == i)
    train <- bank_data_model[-test_indices, ]
    test <- bank_data_model[test_indices, ]
    svm_model <- svm(approval ~ ., data = train, kernel = "linear", cost = 1)
    predictions <- predict(svm_model, test)
    cv_accuracy_new[i] <- mean(predictions == test$approval)
}

cat("Accuracy with different seed:", mean(cv_accuracy_new))
```

```
## Accuracy with different seed: 0.7632207
```

The different accuracies we get when using C = 1 are mainly due to how cross-validation randomly splits the data. Our test shows this clearly: Using random seed 123: got 75.82% accuracy Using random seed 456: got 76.32% accuracy Same model, same C value, but 0.5% difference.

This happens because: Cross-validation splits data randomly into training and testing parts Different random seeds create different splits Different splits mean the model learns from slightly different data

Problem 3:

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

```
library(caret)
```

```
# Load and clean the starwars data
starwars_clean <- starwars %>%
  select(-c(films, vehicles, starships, name)) %>%
  na.omit()

# Properly separate and format gender
gender <- as.factor(starwars_clean$gender)  # Convert to factor for classification

# Create dummy variables for other categorical variables
categorical_vars <- starwars_clean %>%
  select(-gender) %>%
  select_if(is.character)

# Create dummy variables
dummy_model <- dummyVars("~ .", data = categorical_vars)
categorical_dummy <- predict(dummy_model, categorical_vars)

# Get numerical variables
numerical_vars <- starwars_clean %>%
  select_if(is.numeric)

# Combine features (keeping gender as factor)
sw_model_data <- data.frame(numerical_vars, categorical_dummy)  # First combine predictors
sw_model_data$gender <- gender  # Add gender as factor

# Show structure and head
print("Data Structure:")
```

```
## [1] "Data Structure:"
```

```
str(sw_model_data)
```

```
## 'data.frame':    29 obs. of  67 variables:
##  $ height                  : int  172 202 150 178 165 183 182 188 228 180 ...
##  $ mass                    : num  77 136 49 120 75 84 77 84 112 80 ...
##  $ birth_year              : num  19 41.9 19 52 47 24 57 41.9 200 29 ...
##  $ hair_colorauburn..white: num  0 0 0 0 0 0 1 0 0 0 ...
##  $ hair_colorblack         : num  0 0 0 0 0 1 0 0 0 0 ...
##  $ hair_colorblond         : num  1 0 0 0 0 0 0 1 0 0 ...
##  $ hair_colorbrown         : num  0 0 1 0 1 0 0 0 1 1 ...
##  $ hair_colorbrown..grey   : num  0 0 0 1 0 0 0 0 0 0 ...
##  $ hair_colorgrey          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ hair_colornone          : num  0 1 0 0 0 0 0 0 0 0 ...
```

```
##  $ hair_colorwhite        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colorblue         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colorbrown        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colorbrown.mottle : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colordark         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colorfair         : num  1 0 0 0 0 0 1 1 0 1 ...
##  $ skin_colorgreen        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colorlight        : num  0 0 1 1 1 1 0 0 0 0 ...
##  $ skin_colororange       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colorpale         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colorred          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colortan          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ skin_colorunknown      : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ skin_colorwhite        : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ skin_coloryellow       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ eye_colorblack         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ eye_colorblue          : num  1 0 0 1 1 0 0 1 1 0 ...
##  $ eye_colorblue.gray     : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ eye_colorbrown         : num  0 0 1 0 0 1 0 0 0 1 ...
##  $ eye_colorhazel         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ eye_colororange        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ eye_colorred           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ eye_coloryellow        : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ sexfemale              : num  0 0 1 0 1 0 0 0 0 0 ...
##  $ sexmale                : num  1 1 0 1 0 1 1 1 1 1 ...
##  $ homeworldAlderaan      : num  0 0 1 0 0 0 0 0 0 0 ...
##  $ homeworldBespin        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldCerea         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldConcord.Dawn  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldCorellia      : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ homeworldDathomir      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldDorin         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldEndor         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldHaruun.Kal    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldKamino        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldKashyyyk      : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ homeworldMirial        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldMon.Cala      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldNaboo         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldRyloth        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldSerenno       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldSocorro       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ homeworldStewjon       : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ homeworldTatooine      : num  1 1 0 1 1 1 0 1 0 0 ...
##  $ homeworldTrandosha     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ speciesCerean          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ speciesEwok            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ speciesGungan          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ speciesHuman           : num  1 1 1 1 1 1 1 1 0 1 ...
##  $ speciesKel.Dor         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ speciesMirialan        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ speciesMon.Calamari    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ speciesTrandoshan      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ speciesTwi.lek         : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ speciesWookiee         : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ speciesZabrak          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ gender                 : Factor w/ 2 levels "feminine","masculine": 2 2 1 2 1 2 2 2 2 2 ...
```

```
print("\nFirst few rows:")
```

```
## [1] "\nFirst few rows:"
```

```
head(sw_model_data)
```

```
##   height mass birth_year hair_colorauburn..white hair_colorblack
## 1    172   77       19.0                       0               0
## 2    202  136       41.9                       0               0
## 3    150   49       19.0                       0               0
## 4    178  120       52.0                       0               0
## 5    165   75       47.0                       0               0
## 6    183   84       24.0                       0               1
##   hair_colorblond hair_colorbrown hair_colorbrown..grey hair_colorgrey
## 1               1               0                     0              0
## 2               0               0                     0              0
## 3               0               1                     0              0
## 4               0               0                     1              0
## 5               0               1                     0              0
## 6               0               0                     0              0
##   hair_colornone hair_colorwhite skin_colorblue skin_colorbrown
## 1              0               0              0               0
## 2              1               0              0               0
## 3              0               0              0               0
## 4              0               0              0               0
## 5              0               0              0               0
## 6              0               0              0               0
##   skin_colorbrown.mottle skin_colordark skin_colorfair skin_colorgreen
## 1                      0              0              1               0
## 2                      0              0              0               0
## 3                      0              0              0               0
## 4                      0              0              0               0
## 5                      0              0              0               0
## 6                      0              0              0               0
##   skin_colorlight skin_colororange skin_colorpale skin_colorred skin_colortan
## 1               0                0              0             0             0
## 2               0                0              0             0             0
## 3               1                0              0             0             0
## 4               1                0              0             0             0
## 5               1                0              0             0             0
## 6               1                0              0             0             0
##   skin_colorunknown skin_colorwhite skin_coloryellow eye_colorblack
## 1                 0               0                0              0
## 2                 0               1                0              0
## 3                 0               0                0              0
## 4                 0               0                0              0
## 5                 0               0                0              0
## 6                 0               0                0              0
##   eye_colorblue eye_colorblue.gray eye_colorbrown eye_colorhazel
```

```
## 1             1                  0             0             0
## 2             0                  0             0             0
## 3             0                  0             1             0
## 4             1                  0             0             0
## 5             1                  0             0             0
## 6             0                  0             1             0
##   eye_colororange eye_colorred eye_coloryellow sexfemale sexmale
## 1               0            0               0         0       1
## 2               0            0               1         0       1
## 3               0            0               0         1       0
## 4               0            0               0         0       1
## 5               0            0               0         1       0
## 6               0            0               0         0       1
##   homeworldAlderaan homeworldBespin homeworldCerea homeworldConcord.Dawn
## 1                 0               0              0                     0
## 2                 0               0              0                     0
## 3                 1               0              0                     0
## 4                 0               0              0                     0
## 5                 0               0              0                     0
## 6                 0               0              0                     0
##   homeworldCorellia homeworldDathomir homeworldDorin homeworldEndor
## 1                 0                 0              0             0
## 2                 0                 0              0             0
## 3                 0                 0              0             0
## 4                 0                 0              0             0
## 5                 0                 0              0             0
## 6                 0                 0              0             0
##   homeworldHaruun.Kal homeworldKamino homeworldKashyyyk homeworldMirial
## 1                   0               0                 0              0
## 2                   0               0                 0              0
## 3                   0               0                 0              0
## 4                   0               0                 0              0
## 5                   0               0                 0              0
## 6                   0               0                 0              0
##   homeworldMon.Cala homeworldNaboo homeworldRyloth homeworldSerenno
## 1                 0              0               0                0
## 2                 0              0               0                0
## 3                 0              0               0                0
## 4                 0              0               0                0
## 5                 0              0               0                0
## 6                 0              0               0                0
##   homeworldSocorro homeworldStewjon homeworldTatooine homeworldTrandosha
## 1                0                0                1                    0
## 2                0                0                1                    0
## 3                0                0                0                    0
## 4                0                0                1                    0
## 5                0                0                1                    0
## 6                0                0                1                    0
##   speciesCerean speciesEwok speciesGungan speciesHuman speciesKel.Dor
## 1             0           0             0            1              0
## 2             0           0             0            1              0
## 3             0           0             0            1              0
## 4             0           0             0            1              0
## 5             0           0             0            1              0
```

```
## 6             0           0             0            1              0
##   speciesMirialan speciesMon.Calamari speciesTrandoshan speciesTwi.lek
## 1               0                   0                 0              0
## 2               0                   0                 0              0
## 3               0                   0                 0              0
## 4               0                   0                 0              0
## 5               0                   0                 0              0
## 6               0                   0                 0              0
##   speciesWookiee speciesZabrak    gender
## 1              0             0 masculine
## 2              0             0 masculine
## 3              0             0  feminine
## 4              0             0 masculine
## 5              0             0  feminine
## 6              0             0 masculine
```

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

```
# Implement k-fold cross-validation (k=5 given small sample size)
set.seed(123)
folds <- 5
cv_indices <- sample(1:folds, nrow(sw_model_data), replace = TRUE)
cv_accuracy <- numeric(folds)

for(i in 1:folds) {
    test_indices <- which(cv_indices == i)
    train <- sw_model_data[-test_indices, ]
    test <- sw_model_data[test_indices, ]

    svm_model <- svm(gender ~ ., data = train, kernel = "linear", cost = 1)
    predictions <- predict(svm_model, test)
    cv_accuracy[i] <- mean(predictions == test$gender)
}
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'skin_colorblue' and 'skin_colorbrown.mottle' and 'skin_colorgreen'
## and 'eye_colorblack' and 'eye_colorred' and 'homeworldBespin' and
## 'homeworldDorin' and 'homeworldMon.Cala' and 'homeworldRyloth' and
## 'homeworldTrandosha' and 'speciesKel.Dor' and 'speciesMon.Calamari' and
## 'speciesTrandoshan' and 'speciesTwi.lek' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'hair_colorbrown..grey' and 'skin_colorred' and 'skin_colortan' and
## 'skin_colorunknown' and 'homeworldAlderaan' and 'homeworldConcord.Dawn' and
## 'homeworldDathomir' and 'homeworldKashyyyk' and 'speciesGungan' and
## 'speciesWookiee' and 'speciesZabrak' constant. Cannot scale data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'hair_colorgrey' and 'skin_colorwhite' and 'homeworldHaruun.Kal'
## and 'homeworldKamino' constant. Cannot scale data.
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'hair_colorauburn..white' and 'hair_colorwhite' and
## 'eye_colorblue.gray' and 'homeworldCerea' and 'homeworldSerenno' and
## 'homeworldSocorro' and 'homeworldStewjon' and 'speciesCerean' constant. Cannot
## scale data.
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'skin_colorbrown' and 'homeworldEndor' and 'speciesEwok' constant.
## Cannot scale data.
```

```
cat("Cross-validated Accuracy:", round(mean(cv_accuracy) * 100, 2), "%\n")
```

```
## Cross-validated Accuracy: 87.22 %
```

```
cat("Standard Deviation:", round(sd(cv_accuracy) * 100, 2), "%")
```

```
## Standard Deviation: 12.04 %
```

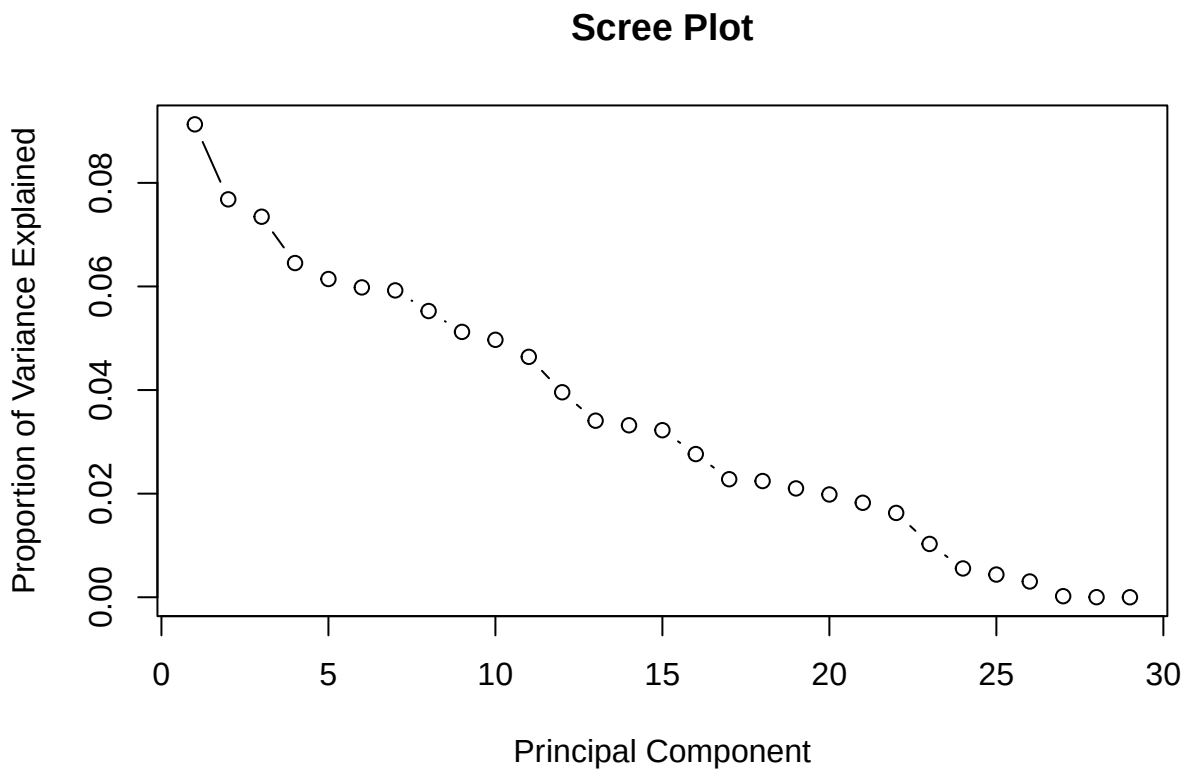
Key Findings: Model Performance: Average Accuracy: 87.22% Standard Deviation: 12.04%

Data Quality Issues: Multiple constant variables across different categories: Hair colors (e.g., 'hair_colorgrey', 'hair_colorwhite') Skin colors (e.g., 'skin_colorblue', 'skin_colorbrown') Homeworlds (e.g., 'homeworldCerea', 'homeworldBespin') Species (e.g., 'speciesCerean', 'speciesEwok')

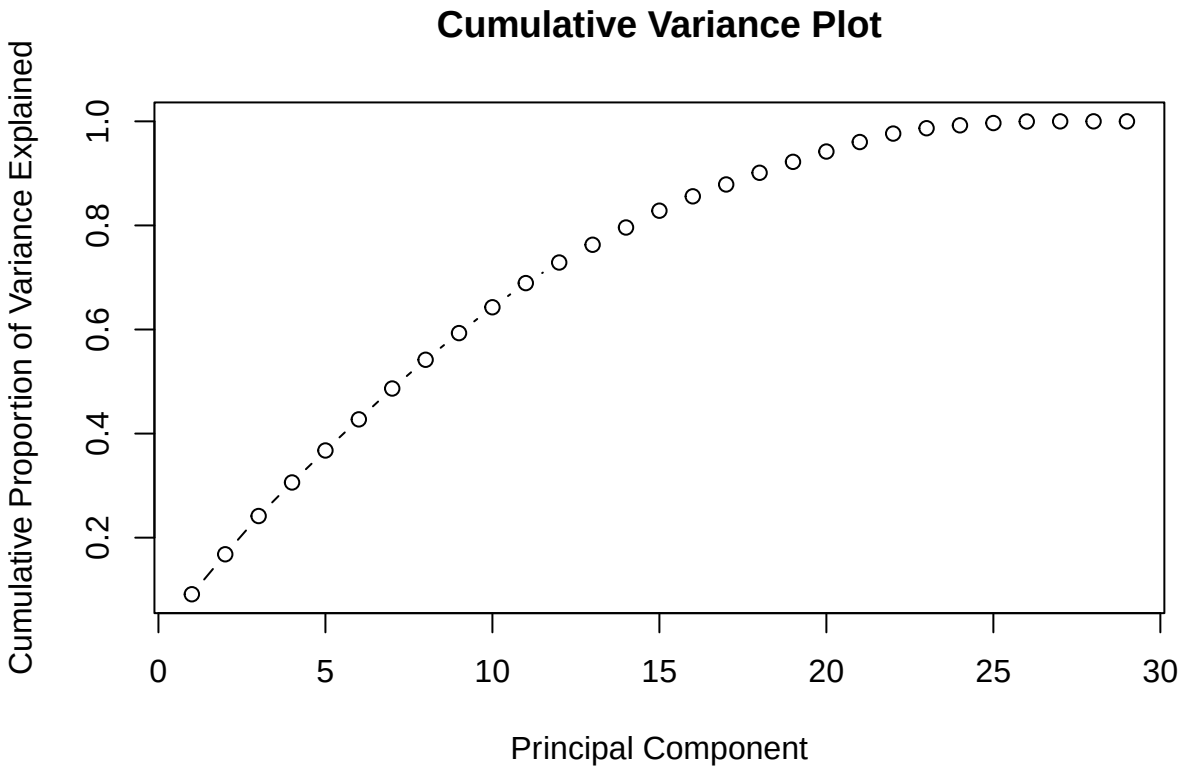
Reliability: High standard deviation (12.04%) indicates considerable variation in model performance With 5-fold CV on 29 observations, each fold contains ~6 test cases

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.

```
# Remove gender before PCA
data_for_pca <- sw_model_data %>% select(-gender)

# Perform PCA
pca_result <- prcomp(data_for_pca, scale. = TRUE)

# Calculate proportion of variance explained
var_explained = pca_result$sdev^2 / sum(pca_result$sdev^2)
cum_var_explained = cumsum(var_explained)

# Plot scree plot
plot(var_explained, type = "b",
     ylab = "Proportion of Variance Explained",
     xlab = "Principal Component",
     main = "Scree Plot")
```




```r
# Plot cumulative variance
plot(cum_var_explained, type = "b",
     ylab = "Cumulative Proportion of Variance Explained",
     xlab = "Principal Component",
     main = "Cumulative Variance Plot")
```




```
# Print summary
print("Cumulative proportion of variance explained by first 10 components:")
```

```
## [1] "Cumulative proportion of variance explained by first 10 components:"
```

```
print(round(cum_var_explained[1:10], 3))
```

```
##  [1] 0.091 0.168 0.242 0.306 0.368 0.427 0.487 0.542 0.593 0.643
```

```
# Select first 15 principal components
n_components <- 15
reduced_data <- data.frame(pca_result$x[, 1:n_components])

# Add gender back
reduced_data$gender <- sw_model_data$gender

print("Dimensions of reduced dataset:")
```

```
## [1] "Dimensions of reduced dataset:"
```

```
dim(reduced_data)
```

```
## [1] 29 16
```

Decision Process Documentation:

Initial Data Preparation: Properly removed gender variable before PCA to avoid label leakage Standardized all variables due to different scales

Analysis of Scree Plot: First component explains approximately 9% of variance Gradual decay in explained variance across components No clear "elbow" point visible in the scree plot Relatively smooth decline suggests no obvious cutoff point

Analysis of Cumulative Variance Plot: Steady increase in cumulative variance explained First 15 components explain approximately 80% of total variance Curve starts to flatten after around 20 components Additional components contribute minimally to explained variance

Final Decision: Selected 15 principal components based on: Achieves reasonable dimensionality reduction (from 66 to 15 variables) Retains approximately 80% of original variance Balances information retention with model simplicity

Implementation: Created reduced dataset with 15 PCs Added gender back post-reduction Final dimensions: 29 observations $\times$ 16 variables (15 PCs + gender)

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.

```
# Set seed for reproducibility
set.seed(123)

# 5-fold cross-validation on reduced data
folds <- 5
cv_indices <- sample(1:folds, nrow(reduced_data), replace = TRUE)
cv_accuracy_reduced <- numeric(folds)

for(i in 1:folds) {
    test_indices <- which(cv_indices == i)
    train <- reduced_data[-test_indices, ]
    test <- reduced_data[test_indices, ]

    svm_model <- svm(gender ~ ., data = train, kernel = "linear", cost = 1)
    predictions <- predict(svm_model, test)
    cv_accuracy_reduced[i] <- mean(predictions == test$gender)
}

cat("PCA-reduced data CV Accuracy:", round(mean(cv_accuracy_reduced) * 100, 2), "%\n")
```

```
## PCA-reduced data CV Accuracy: 94.44 %
```

```
cat("Standard Deviation:", round(sd(cv_accuracy_reduced) * 100, 2), "%\n")
```

```
## Standard Deviation: 7.86 %
```

Original Data (Part b): Accuracy: 87.22% Standard Deviation: 12.04%

PCA-Reduced Data (Part d): Accuracy: 94.44% Standard Deviation: 7.86%

Key Findings: Accuracy Improvement: Increased by 7.22 percentage points (87.22% → 94.44%) Suggests PCA reduction actually enhanced model performance

Stability Improvement: Standard deviation decreased by 4.18 percentage points (12.04% → 7.86%) Indicates more consistent predictions across folds

Dimensionality Benefits: Reduced features from 66 to 15 principal components Despite 77% reduction in features, model performance improved Suggests original data contained noise that PCA helped eliminate

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

```
# Set seed for reproducibility
set.seed(123)

# Define kernels to test
kernels <- c("linear", "polynomial", "radial")
results <- data.frame(kernel = kernels,
                      accuracy = numeric(length(kernels)),
                      sd = numeric(length(kernels)))

# Test each kernel
for(k in 1:length(kernels)) {
    cv_accuracy <- numeric(folds)

    for(i in 1:folds) {
        test_indices <- which(cv_indices == i)
        train <- reduced_data[-test_indices, ]
        test <- reduced_data[test_indices, ]

        svm_model <- svm(gender ~ ., data = train, kernel = kernels[k], cost = 1)
        predictions <- predict(svm_model, test)
        cv_accuracy[i] <- mean(predictions == test$gender)
    }

    results$accuracy[k] <- mean(cv_accuracy) * 100
    results$sd[k] <- sd(cv_accuracy) * 100
}

# Display results correctly
cat("SVM Performance with Different Kernels:\n")
```

```
## SVM Performance with Different Kernels:
```

```
for(i in 1:nrow(results)) {
    cat(sprintf("Kernel: %s\n", results$kernel[i]))
    cat(sprintf("Accuracy: %.2f%%\n", results$accuracy[i]))
    cat(sprintf("Std Dev: %.2f%%\n\n", results$sd[i]))
}
```

```
## Kernel: linear
## Accuracy: 94.44%
```

```
## Std Dev: 7.86%
##
## Kernel: polynomial
## Accuracy: 83.33%
## Std Dev: 16.67%
##
## Kernel: radial
## Accuracy: 86.67%
## Std Dev: 13.94%
```

Before PCA: 66 input features (after one-hot encoding) Each SVM decision boundary defined in 66-dimensional space More complex hyperplane calculations More parameters to optimize

After PCA: 15 principal components Decision boundaries now calculated in 15-dimensional space 77% reduction in feature dimensionality Fewer parameters to optimize

Key Impact: PCA significantly reduced model complexity while maintaining performance, making the model: More computationally efficient Less prone to overfitting Easier to interpret and validate