Machine Learning Final Project

David Costa, Lucas Gaspar

2025-04-08

Project Description

The goal of this project is to apply multiple machine/statistical learning techniques we've learned over the course of the semester in MATH 4050 to a fictional dataset of 8,000 observations with 31 features. We intend to use unsupervised learning (Hierarchical clustering) to conduct some preliminary examination and help guide the decision of which features to use as response variables (1 categorical and 1 numerical). We will then use best practices to develop Linear/Logistic regression, Decision Tree, Random Forest, and Support Vector Machine models. Along the way we will use various tuning methods, validation techiniques, and dimensional reduction techniques to ensure that our models are well fit, and robust.

The data used in the project is a fictional dataset containing 31 features about a given household. It contains 16 categorical and 15 numerical variables:

Feature	Description
urbrur	Whether the household is located in urban or rural location
hhsize	Household size (number of members)
statocc	Whether the household rents, owns, or has free occupancy
rooms	Number of rooms in the household
bedrooms	Number of bedrooms in the household
floor	Floor number where the household is located (if applicable)
walls	Type of walls in the household
roof	Type of roof in the household
electricity	Whether the household has electricity
cook_fuel	Type of cooking fuel used in the household
phone	Whether the household has a phone
cell	Whether the household has a mobile phone
car	Whether the household has a car
bicycle	Whether the household has a bicycle
motorcycle	Whether the household has a motorcycle
refrigerator	Whether the household has a refrigerator
tv	Whether the household has a television
radio	Whether the household has a radio
bank	Whether the household has a bank account
exp_01	Annual spending on Food and non-alcoholic beverages
exp_02	Annual spending on Alcoholic beverages, tobacco and narcotics
exp_03	Annual spending on Clothing and footwear
exp_04	Annual spending on Housing, water, electricity, gas and other fuels
exp_05	Annual spending on Furnishing, household equipment and routine maintenance
exp_06	Annual spending on Health
exp_07	Annual spending on Transport
exp_08	Annual spending on Communication

Feature	Description
exp__09	Annual spending on Recreation and culture
exp__10	Annual spending on Education
exp__11	Annual spending on Catering and accommodation services
exp__12	Annual spending on Miscellaneous goods and services

```
df <- read.csv("https://raw.githubusercontent.com/costad3atwit/Machine_Learning_Final/refs/heads/main/p
```

```
#install.packages("gower")
library(gower)
#install.packages("StatMatch")
library(StatMatch)
```

Examining and Making the Dataset Usable

```
#Rename confusing variables

df <- df %>% rename(
  spend_food = exp_01,
  spend_alcohol = exp_02,
  spend_clothes = exp_03,
  spend_housing = exp_04,
  spend_furnishing = exp_05,
  spend_health = exp_06,
  spend_transport = exp_07,
  spend_communication = exp_08,
  spend_recreation = exp_09,
  spend_education = exp_10,
  spend_catering = exp_11,
  spend_misc = exp_12
)

df$X <- NULL
```

```
df$tv <- as.factor(df$tv)
df$urbrur <- as.factor(df$urbrur)
df$statocc <- as.factor(df$statocc)
df$floor <- as.factor(df$floor)
df$walls <- as.factor(df$walls)
df$roof <- as.factor(df$roof)
df$electricity <- as.factor(df$electricity)
df$cook_fuel <- as.factor(df$cook_fuel)
df$phone <- as.factor(df$phone)
df$cell <- as.factor(df$cell)
df$car <- as.factor(df$car)
df$bicycle <- as.factor(df$bicycle)
df$motorcycle <- as.factor(df$motorcycle)
df$refrigerator <- as.factor(df$refrigerator)
df$radio <- as.factor(df$radio)
df$bank <- as.factor(df$bank)
```

```r
str(df)
```

```r
head(df)
```

```
##   urbrur hhsize statocc rooms bedrooms  floor walls     roof electricity
## 1  Urban      1   Owned     1        1 Cement Brick    Metal         Yes
## 2  Urban      1  Rented     1        0 Cement Stone    Metal         Yes
## 3  Urban      2   Owned     4        1 Cement Brick Concrete         Yes
## 4  Urban      2   Owned     1        1 Cement Brick    Metal         Yes
## 5  Urban      1  Rented     3        2  Other Brick Concrete         Yes
## 6  Urban      2   Owned     3        2 Cement Brick Concrete         Yes
##     cook_fuel phone cell car bicycle motorcycle refrigerator  tv radio bank
## 1         Gas    No   No  No      No         No          Yes Yes    No   No
## 2   Petroleum    No  Yes  No      No         No           No  No    No  Yes
## 3 Electricity    No  Yes  No      No         No          Yes Yes   Yes  Yes
## 4         Gas    No  Yes  No      No         No          Yes Yes    No  Yes
## 5        Wood   Yes  Yes Yes      No         No          Yes Yes    No  Yes
## 6         Gas   Yes   No  No      No         No          Yes Yes   Yes   No
##   spend_food spend_alcohol spend_clothes spend_housing spend_furnishing
## 1       1279           162           266          1662              153
## 2       1755            71           279           676              258
## 3       1722            59           696          3724              390
## 4       3700           174           660          2921              340
## 5       1492            82           603          5556              957
## 6       2433            15           152          3900              286
##   spend_health spend_transport spend_communication spend_recreation
## 1           63             372                 103               76
## 2           78             273                 264              103
## 3          188             710                 582              389
## 4          238            1049                 592              206
## 5          165            2386                 516              441
## 6          331             210                 660              112
##   spend_education spend_catering spend_misc
## 1              38            544        326
## 2              90            571        234
## 3             356            305        836
## 4             149            576        928
## 5             267           1395        689
## 6              31             45        347
```

Unsupervised Data Analysis: Factor Analysis of Mixed Data

FAMD allows us to look at our data in a new way by significantly reducing its dimensionality. It differs from PCA in the fact that it can be used on mixed data, containing both quantitative variables as well as qualitative ones. This unsupervised process can help with better understanding trends in the data but also makes it significantly harder to draw conclusions as variables become distorted.

Step 1: Perform FAMD analysis

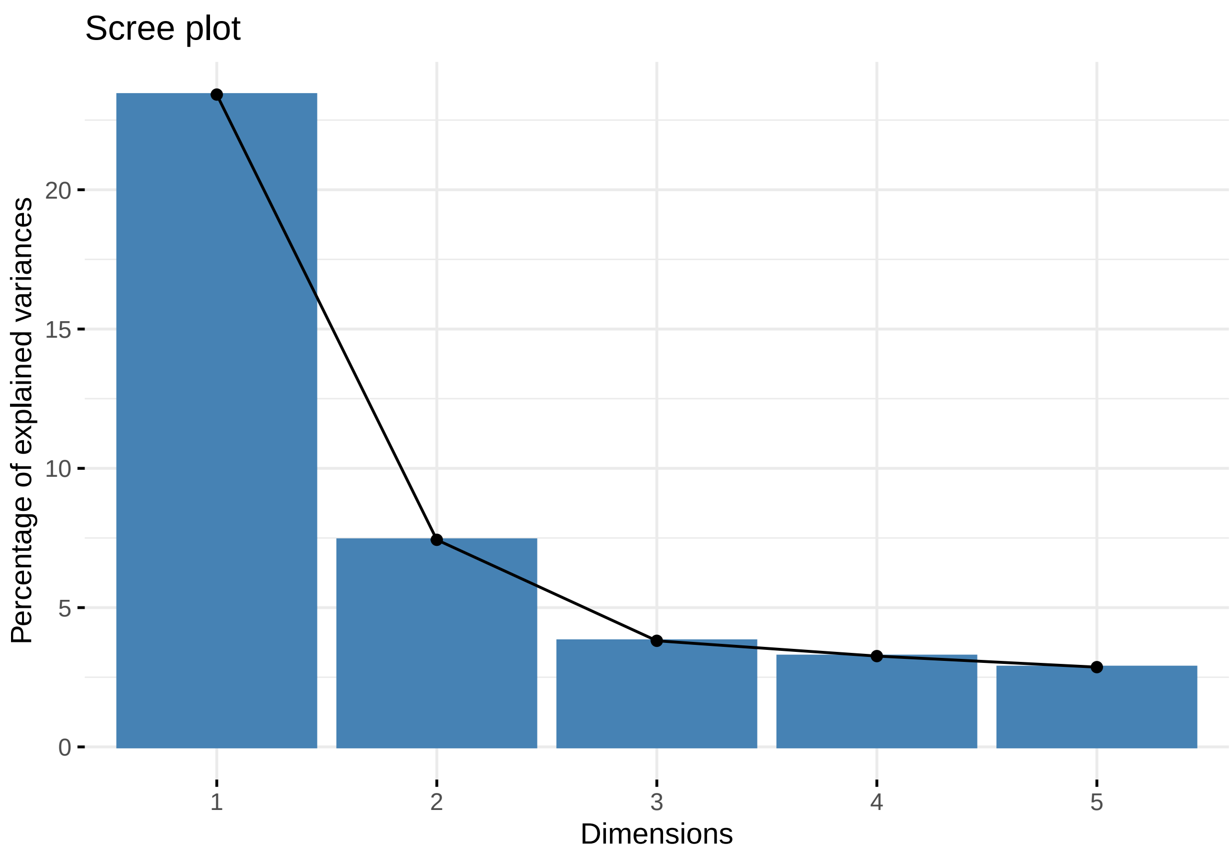
First, we load the packages and create a new dataframe so we don't alter our original one we will continue using for other techniques. Then, we perform famd, this automatically normalizes all our features and analyzes their relationships. Essentially, this is performing PCA on the quantitative features, and MCA on the qualitative ones.

```
library(FactoMineR)
library(factoextra)

df_famd <- df

res.famd <- FAMD(df_famd, graph = FALSE, ncp = 5)
```

Step 2: Create Plots

We can now visualize this data in significantly fewer dimensions thanks to our new Dim1 and Dim2. This allows for extremely useful plotting.

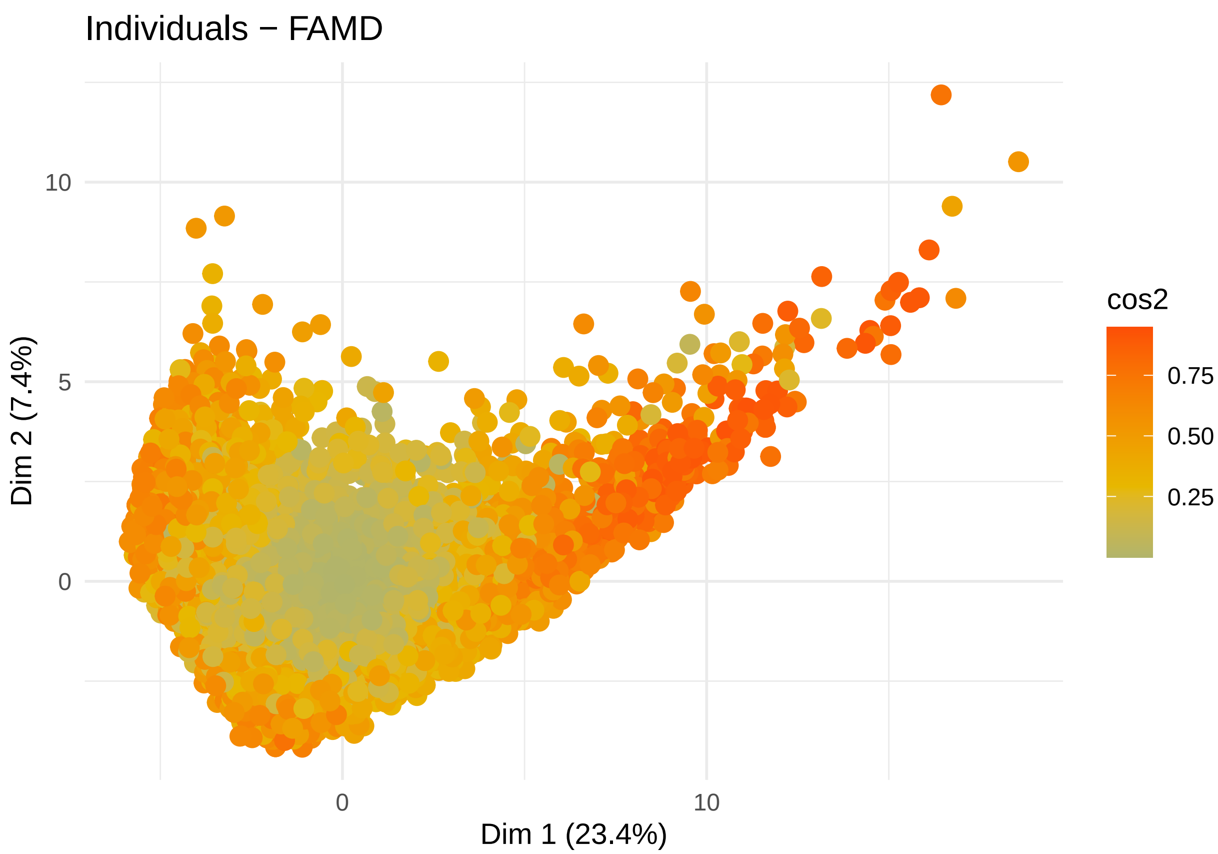
```
#Scree
scree_plot <- fviz_screeplot(res.famd)
print(scree_plot)
```



Scree Plot

- represents how much variance explained in how many dimensions. ~30% in 2 dimensions is not very good, indicating that we there may be many variables that are have great impact.
- The sharp drop does indicate that these two are significantly more important than others.

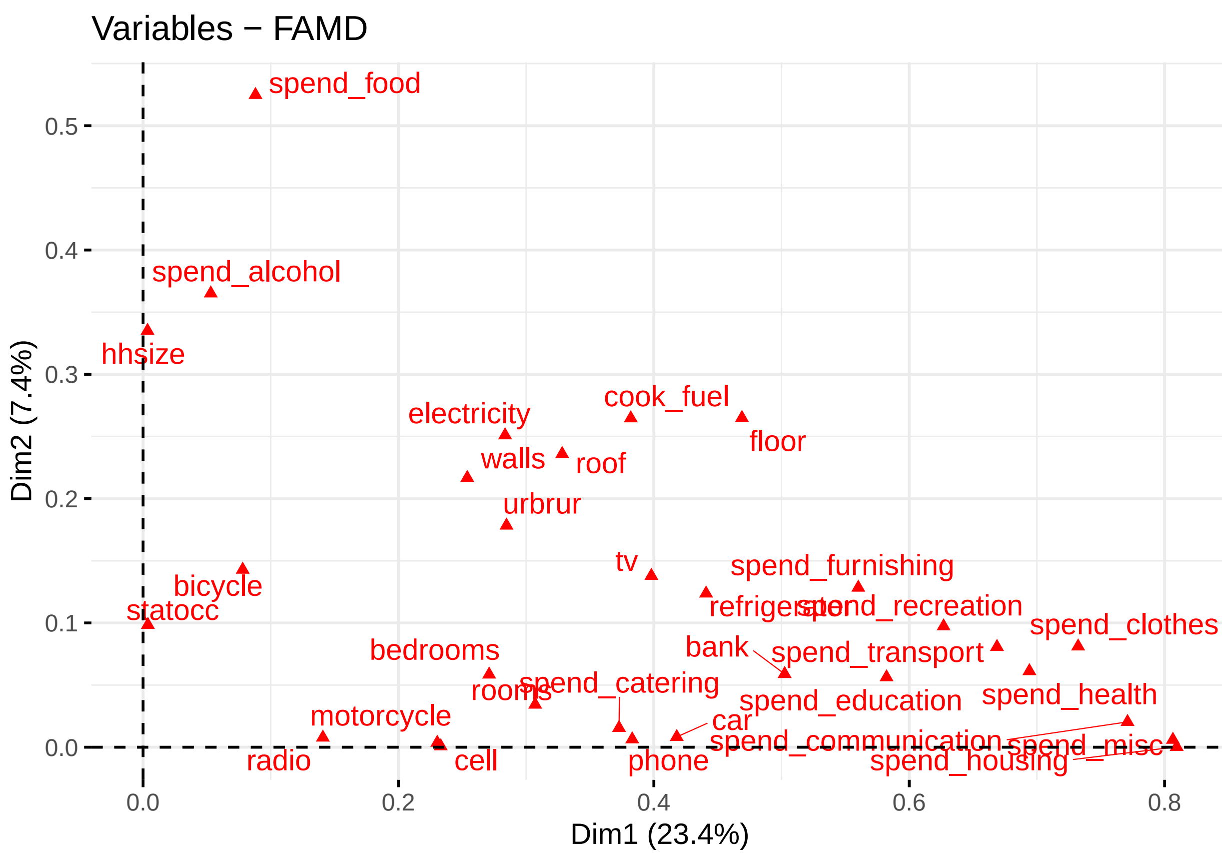
```r
# Individual
ind_coords <- as.data.frame(res.famd$ind$coord)

library(ggplot2)

# Get the cos2 values for coloring
cos2 <- rowSums(res.famd$ind$cos2[,1:2])

# df for plotting
plot_df <- data.frame(
Dim1 = ind_coords[,1],
  Dim2 = ind_coords[,2],
  cos2 = cos2
)

# Create the plot
ind_plot <- ggplot(plot_df, aes(x = Dim1, y = Dim2, color = cos2)) +
  geom_point(size = 3) +
  scale_color_gradient2(low = "#00AFBB", mid = "#E7B800", high = "#FC4E07",
                        midpoint = median(cos2)) +
  theme_minimal() +
  labs(title = "Individuals - FAMD",
       x = paste0("Dim 1 (", round(res.famd$eig[1,2], 1), "%)"),
       y = paste0("Dim 2 (", round(res.famd$eig[2,2], 1), "%)"))

print(ind_plot)
```



Individuals Plot

- plots our altered data in the two new dimensions.
- warmer colors indicate a better fit in this new space.
- We can see a V shape where the data is better fit, indicating two groups that we can analyze.

```r
# Variable
var_plot <- fviz_famd_var(res.famd, repel = TRUE)
print(var_plot)
```



Variables

- looks at our original, unaltered variables and how they exist in the new dimensions.
- The further the variable along an axis, the stronger it represents that dimension.
- We can see patterns along each dimension such as:
  - Dim1 seems to indicate more frugal spending which can relate to a persons overall wealth or spending habits.
  - Dim 2 seems to indicate variables that are short term such as food and alcohol

While we did not use FAMD to reduce the dimensionality for the rest of our analysis, it still provided insight to how we can look at our variables and what correlations there may be.

Unsupervised Data Analysis: Clustering

We use cluster analysis to help pick an interesting response variable via PAM clustering

Since we are looking at mixed data, we have to use Gower distance formula to calculate the distance between different data entries. This formula can be show as:

$$d_{ij} = \frac{\sum_{k=1}^{p} w_k \delta_{ijk}}{\sum_{k=1}^{p} w_k}$$

The $w_k \delta_{ijk}$ represents different formulas used for different types of data:

Numeric:

$$\delta_{ijk} = \frac{|x_{ik} - x_{jk}|}{R_k}$$

Categorical:

$$\delta_{ijk} = \begin{cases} 0 & \text{if } x_{ik} = x_{jk} \\ 1 & \text{if } x_{ik} \neq x_{jk} \end{cases}$$

Binary:

$$\delta_{ijk} = \begin{cases} 0 & \text{if } x_{ik} = x_{jk} = 0 \text{ or } x_{ik} = x_{jk} = 1 \\ 1 & \text{if } x_{ik} \neq x_{jk} \end{cases}$$

By using all of these together, we can form new interpretation of distance to measure our point.

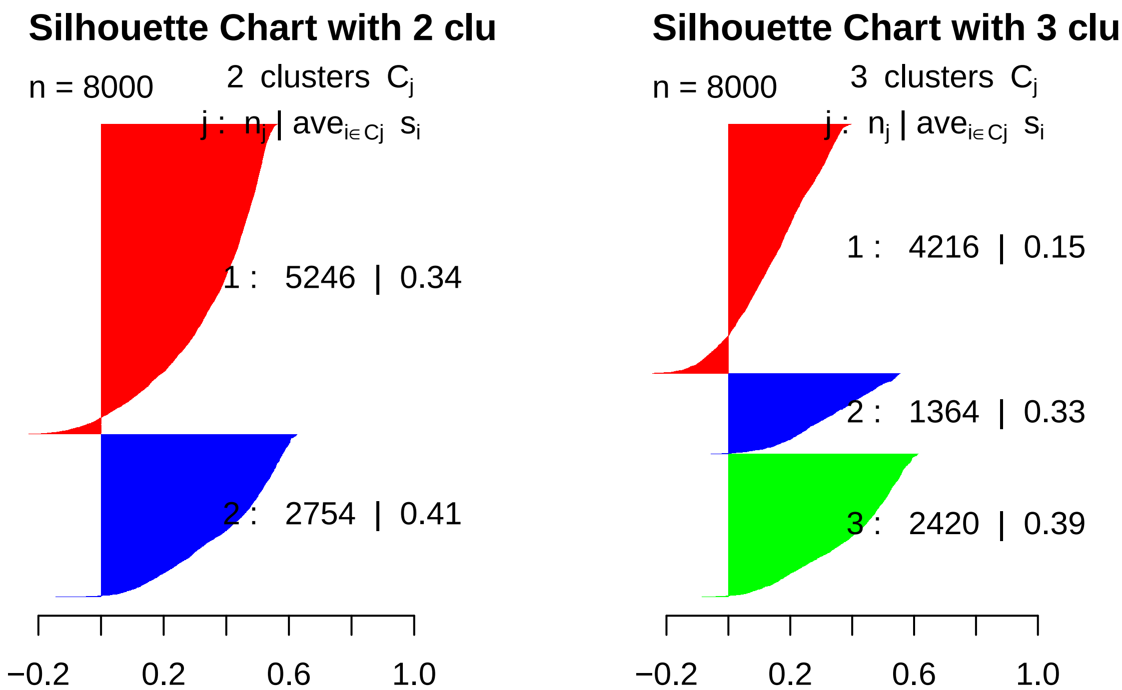
```
gower_df = gower.dist(df)
```

```
library(cluster)
```

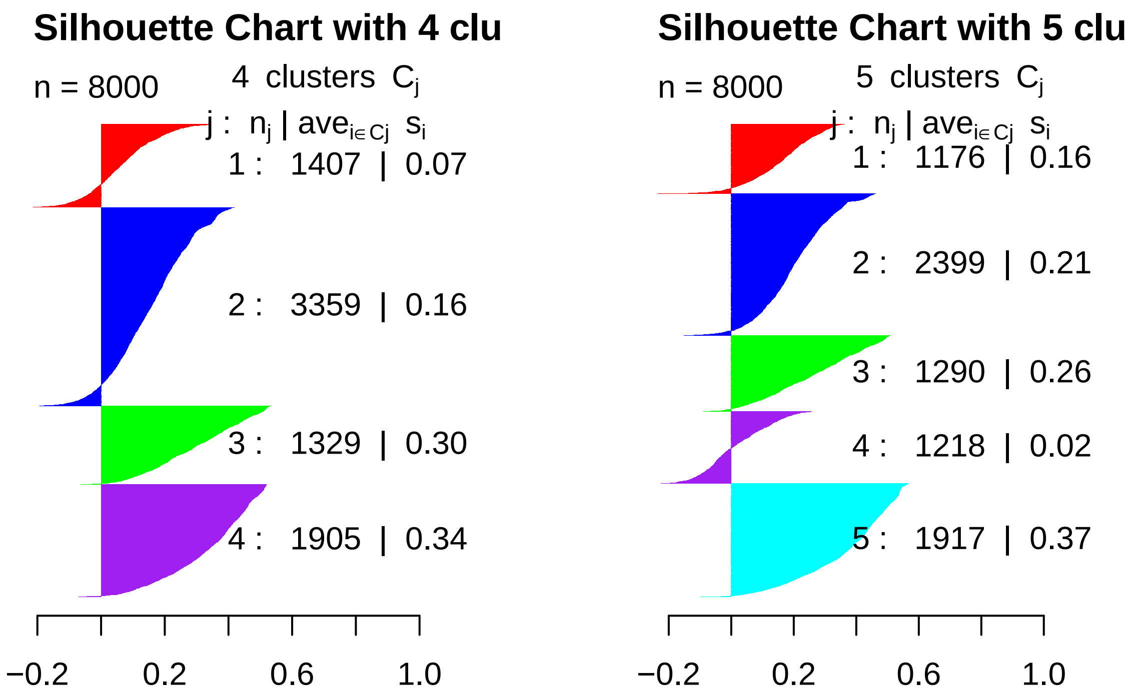
We'll use the Partitions Around Mediods (PAM) Method of clustering to look for best clustering and subsequently the most impactful features. We'll use those features to decide what to predict.

```
pam_result_2 <- pam(gower_df, k=2, diss=TRUE)
sil_2 = silhouette(pam_result_2)
pam_result_3 <- pam(gower_df, k=3, diss=TRUE)
sil_3 = silhouette(pam_result_3)
pam_result_4 <- pam(gower_df, k=4, diss=TRUE)
sil_4 = silhouette(pam_result_4)
pam_result_5 <- pam(gower_df, k=5, diss=TRUE)
sil_5 = silhouette(pam_result_5)
pam_result_6 <- pam(gower_df, k=6, diss=TRUE)
sil_6 = silhouette(pam_result_6)
pam_result_7 <- pam(gower_df, k=7, diss=TRUE)
sil_7 = silhouette(pam_result_7)
```

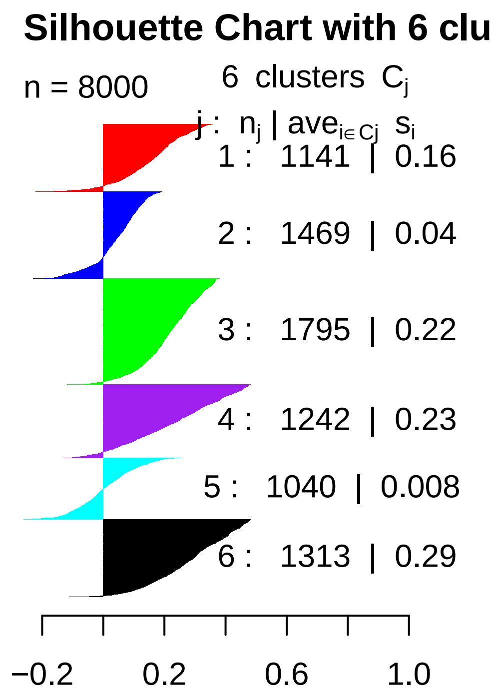
```
par(mfrow = c(1,2))
plot(sil_2, main = "Silhouette Chart with 2 clusters",
     xlab = "", sub = "", cex = 0.9,
     col = c("red", "blue"),  # Add colors for different clusters
     border = NA,             # Remove borders around individual lines
     do.n.k = TRUE,           # Show number of observations and clusters
     do.clus.stat = TRUE)     # Show cluster statistics
plot(sil_3, main = "Silhouette Chart with 3 clusters",
     xlab = "", sub = "", cex = 0.9,
     col = c("red", "blue","green"),  # Add colors for different clusters
     border = NA,             # Remove borders around individual lines
     do.n.k = TRUE,           # Show number of observations and clusters
     do.clus.stat = TRUE)     # Show cluster statistics
```

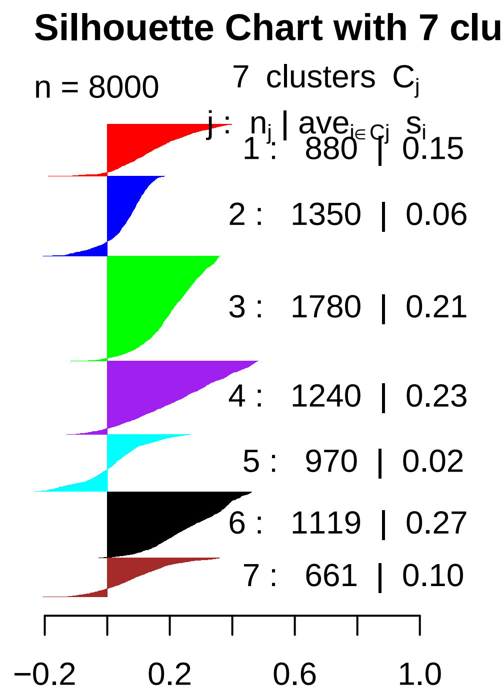



```
par(mfrow = c(1,2))
plot(sil_4, main = "Silhouette Chart with 4 clusters",
     xlab = "", sub = "", cex = 0.9,
     col = c("red", "blue","green", "purple"),  # Add colors for different clusters
     border = NA,              # Remove borders around individual lines
     do.n.k = TRUE,            # Show number of observations and clusters
     do.clus.stat = TRUE)      # Show cluster statistics
plot(sil_5, main = "Silhouette Chart with 5 clusters",
     xlab = "", sub = "", cex = 0.9,
     col = c("red", "blue","green", "purple", "cyan"),  # Add colors for different clusters
     border = NA,              # Remove borders around individual lines
     do.n.k = TRUE,            # Show number of observations and clusters
     do.clus.stat = TRUE)      # Show cluster statistics
```




```
par(mfrow = c(1,2))
plot(sil_6, main = "Silhouette Chart with 6 clusters",
     xlab = "", sub = "", cex = 0.9,
     col = c("red", "blue","green", "purple", "cyan", "black"),  # Add colors for different clusters
     border = NA,              # Remove borders around individual lines
     do.n.k = TRUE,            # Show number of observations and clusters
     do.clus.stat = TRUE)      # Show cluster statistics
plot(sil_7, main = "Silhouette Chart with 7 clusters",
     xlab = "", sub = "", cex = 0.9,
     col = c("red", "blue","green", "purple", "cyan", "black", "brown"),  # Add colors for different cl
     border = NA,              # Remove borders around individual lines
     do.n.k = TRUE,            # Show number of observations and clusters
     do.clus.stat = TRUE)      # Show cluster statistics
```





It looks like our best clustering occurred with only 2 clusters because the average silhouette width was the highest among each cluster there. We'll continue using the 2 cluster model.

```
df$cluster <- pam_result_2$clustering

# Create empty lists to store results
numeric_results <- list()
categorical_results <- list()

# Loop through each variable (except the cluster variable)
for(var_name in names(df)[1:31]) {

  if(is.numeric(df[[var_name]])) {
    # For numeric variables
    # Calculate mean, median, standard deviation by cluster
    stats_by_cluster <- aggregate(df[[var_name]] ~ cluster, data=df,
                                  FUN=function(x) c(mean=mean(x, na.rm=TRUE),
                                                    median=median(x, na.rm=TRUE),
                                                    sd=sd(x, na.rm=TRUE)))

    # Calculate standardized difference (Cohen's d)
    means <- c(stats_by_cluster[1,2][1], stats_by_cluster[2,2][1])
    sds <- c(stats_by_cluster[1,2][3], stats_by_cluster[2,2][3])
    pooled_sd <- sqrt((sds[1]^2 + sds[2]^2)/2)
    effect_size <- abs(means[1] - means[2])/pooled_sd
```

```r
    # Perform t-test between clusters
    t_test_result <- t.test(df[[var_name]] ~ df$cluster)

    # Store results
    numeric_results[[var_name]] <- data.frame(
      variable = var_name,
      cluster1_mean = means[1],
      cluster2_mean = means[2],
      mean_difference = abs(means[1] - means[2]),
      effect_size = effect_size,
      p_value = t_test_result$p.value
    )

  } else {
    # For categorical variables
    # Create contingency table
    cont_table <- table(df$cluster, df[[var_name]])

    # Calculate proportions within each cluster
    prop_table <- prop.table(cont_table, margin=1)

    # Chi-square test
    chi_test <- chisq.test(cont_table)

    # Calculate Cramer's V (effect size for categorical variables)
    n <- sum(cont_table)
    cramers_v <- sqrt(chi_test$statistic / (n * (min(dim(cont_table)) - 1)))

    # Store results
    categorical_results[[var_name]] <- data.frame(
      variable = var_name,
      chi_square = chi_test$statistic,
      p_value = chi_test$p.value,
      cramers_v = cramers_v
    )
  }
}

# Combine results
numeric_df <- do.call(rbind, numeric_results)
categorical_df <- do.call(rbind, categorical_results)

# Sort by effect size/statistical significance
numeric_df <- numeric_df[order(-numeric_df$effect_size),]
categorical_df <- categorical_df[order(-categorical_df$cramers_v),]

# Print top 10 most differentiating variables
print("Top 10 Numeric Variables:")
```

```
## [1] "Top 10 Numeric Variables:"
```

```r
print(head(numeric_df, 10))
```

```
##                                 variable cluster1_mean cluster2_mean
## spend_misc                    spend_misc      945.6231      251.4158
## spend_housing              spend_housing     4097.0349     1153.5062
## spend_communication spend_communication      705.9685      100.2092
## spend_clothes              spend_clothes      826.6533      353.3046
## spend_health                spend_health      368.0526      136.1816
## spend_transport          spend_transport     1605.0799      334.7524
## spend_education          spend_education      782.9264      159.0221
## spend_recreation        spend_recreation      438.1685      124.7549
## spend_catering            spend_catering      437.4249      195.7636
## spend_furnishing        spend_furnishing      681.6548      298.0461
##                     mean_difference effect_size p_value
## spend_misc                 694.2074   1.7658362       0
## spend_housing             2943.5287   1.7590074       0
## spend_communication        605.7594   1.3930675       0
## spend_clothes              473.3486   1.3450662       0
## spend_health               231.8711   1.2068423       0
## spend_transport           1270.3275   1.0985316       0
## spend_education            623.9043   1.0675528       0
## spend_recreation           313.4136   1.0576457       0
## spend_catering             241.6613   0.9019316       0
## spend_furnishing           383.6087   0.8591499       0
```

```
print("Top 10 Categorical Variables:")
```

```
## [1] "Top 10 Categorical Variables:"
```

```
print(head(categorical_df, 10))
```

```
##                  variable chi_square     p_value cramers_v
## bank                 bank   4606.710 0.0000e+00 0.7588404
## tv                     tv   4199.100 0.0000e+00 0.7244912
## refrigerator refrigerator   4000.040 0.0000e+00 0.7071103
## floor               floor   3914.736 0.0000e+00 0.6995298
## cook_fuel       cook_fuel   3822.593 0.0000e+00 0.6912482
## roof                 roof   2681.268 0.0000e+00 0.5789287
## electricity   electricity   2574.615 0.0000e+00 0.5672979
## urbrur             urbrur   2551.173 0.0000e+00 0.5647093
## walls               walls   2276.776 0.0000e+00 0.5334763
## cell                 cell   1453.090 6.1093e-318 0.4261881
```

Based on the top ten most meaningful features derived above through clustering, we've decided to move forward with our supervised learning techniques predicting on **spend_housing** *(spending on housing AND utilities)*, **and tv** *(television)* because their effect size is high and the findings may be interesting

```
#Before we move on we need to remove the cluster variable from the dataframe so we don't skew our other
df$cluster = NULL
```

Subsetting the Data for Regression

We'll need to set aside some test data before we work with any supervised learning techniques so that we can perform accurate validation

```r
sample = sample(nrow(df), nrow(df) * .75)
df.train <- df[sample, ]
df.test <- df[-sample, ]
```

Linear Regression

In this section, we'll develop a linear regression model to predict housing expenditures (`spend_housing`) based on various household characteristics. Linear regression models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data.

The general form of the linear regression equation is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$$

Where: - Y is the response variable (spend_housing) - β_0 is the intercept - $\beta_1, \beta_2, ..., \beta_p$ are the coefficients - $X_1, X_2, ..., X_p$ are the predictor variables - ϵ is the error term

The training process involves minimizing the Mean Squared Error (MSE), which is calculated as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where y_i is the actual value and $\hat{y}_i$ is the predicted value.

Feature Selection using Forward Stepwise Selection

First, we'll apply forward stepwise selection to identify the most important predictors for our model:

```r
# Load necessary libraries for linear regression
library(leaps)
library(caret)

# Set seed for reproducibility
set.seed(123)

# Create a formula excluding the response variable
predictors <- setdiff(names(df), c("spend_housing", "cluster"))
formula_all <- as.formula(paste("spend_housing ~", paste(predictors, collapse = " + ")))

# Null model (for comparison)
null_model <- lm(spend_housing ~ 1, data = df.train)
null_mse_train <- mean(residuals(null_model)^2)
cat("Null model training MSE:", null_mse_train, "\n")
```

```
## Null model training MSE: 5314971
```

```r
# Forward stepwise selection
forward_selection <- suppressWarnings(step(lm(spend_housing ~ 1, data = df.train),
                              direction = "forward",
                              scope = formula_all,
                              trace = FALSE))
```

```r
# Display the selected variables
cat("Variables selected by forward selection:\n")
```

```
## Variables selected by forward selection:
```

```r
print(formula(forward_selection))
```

```
## spend_housing ~ spend_communication + walls + spend_misc + rooms +
##     floor + cell + spend_clothes + spend_catering + spend_health +
##     motorcycle + refrigerator + cook_fuel + roof + statocc +
##     spend_food + spend_alcohol + urbrur + spend_furnishing +
##     spend_recreation + bicycle + hhsize + car + spend_education +
##     electricity + bank + bedrooms
```

Fitting the Linear Model with Selected Features

Now, we'll fit the linear regression model using the features selected by forward stepwise selection:

```r
# Fit the linear model with selected predictors
model_forward <- lm(formula(forward_selection), data = df.train)
summary_forward <- summary(model_forward)

# Display model summary and key statistics
print(summary_forward)
```

```
##
## Call:
## lm(formula = formula(forward_selection), data = df.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4661.6  -352.6   -18.5   314.8  5528.5
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             3.474e+02  2.845e+02   1.221 0.222169
## spend_communication     1.336e+00  3.603e-02  37.088  < 2e-16 ***
## wallsBrick              9.166e+02  3.473e+01  26.388  < 2e-16 ***
## wallsCardboard          1.933e+02  7.866e+01   2.457 0.014033 *
## wallsConcrete          -5.349e+02  5.357e+01  -9.986  < 2e-16 ***
## wallsMetal             -1.082e+02  6.443e+01  -1.679 0.093146 .
## wallsOther             -3.090e+02  6.949e+01  -4.447 8.88e-06 ***
## wallsStone             -1.797e+02  8.835e+01  -2.034 0.042019 *
## wallsWood               2.346e+02  3.787e+01   6.194 6.24e-10 ***
## spend_misc              1.647e+00  5.045e-02  32.653  < 2e-16 ***
## rooms                   1.970e+02  7.668e+00  25.686  < 2e-16 ***
## floorEarth             -2.066e+02  2.837e+01  -7.283 3.68e-13 ***
## floorOther              5.912e+02  2.915e+01  20.282  < 2e-16 ***
## floorTile               3.700e+02  1.581e+02   2.340 0.019296 *
## floorWood              -1.166e+02  1.613e+02  -0.723 0.469510
## cellYes                -4.319e+02  2.369e+01 -18.233  < 2e-16 ***
```

```
## spend_clothes        -1.331e+00  8.834e-02 -15.069  < 2e-16 ***
## spend_catering        1.024e+00  5.013e-02  20.426  < 2e-16 ***
## spend_health          1.508e+00  1.067e-01  14.134  < 2e-16 ***
## motorcycleYes        -3.646e+02  2.337e+01 -15.602  < 2e-16 ***
## refrigeratorYes       2.996e+02  2.606e+01  11.496  < 2e-16 ***
## cook_fuelElectricity  1.222e+01  6.423e+01   0.190 0.849145
## cook_fuelGas         -4.013e+02  4.172e+01  -9.620  < 2e-16 ***
## cook_fuelOther       -4.575e+02  5.953e+01  -7.685 1.77e-14 ***
## cook_fuelPetroleum   -1.754e+02  5.444e+01  -3.221 0.001286 **
## cook_fuelWood        -2.020e+02  3.887e+01  -5.197 2.09e-07 ***
## roofConcrete         -4.776e+00  2.774e+02  -0.017 0.986263
## roofMetal            -3.099e+02  2.773e+02  -1.117 0.263876
## roofOther            -2.237e+02  2.796e+02  -0.800 0.423716
## roofScrap            -1.730e+02  2.832e+02  -0.611 0.541158
## roofSlate            -1.410e+02  2.833e+02  -0.498 0.618654
## roofThatch           -2.553e+02  2.794e+02  -0.914 0.360921
## roofTile             -1.335e+02  3.316e+02  -0.403 0.687198
## roofWood             -2.877e+02  2.836e+02  -1.014 0.310488
## statoccOwned          8.606e+01  2.735e+01   3.146 0.001661 **
## statoccRented        -1.397e+02  3.442e+01  -4.058 5.01e-05 ***
## spend_food            1.119e-01  9.944e-03  11.253  < 2e-16 ***
## spend_alcohol        -1.296e+00  1.441e-01  -8.990  < 2e-16 ***
## urbrurUrban           2.450e+02  2.511e+01   9.758  < 2e-16 ***
## spend_furnishing      7.493e-01  5.566e-02  13.463  < 2e-16 ***
## spend_recreation     -9.127e-01  7.235e-02 -12.616  < 2e-16 ***
## bicycleYes           -1.599e+02  2.617e+01  -6.110 1.06e-09 ***
## hhsize               -3.369e+01  6.545e+00  -5.147 2.73e-07 ***
## carYes               -1.238e+02  2.744e+01  -4.512 6.54e-06 ***
## spend_education       1.146e-01  3.061e-02   3.744 0.000183 ***
## electricityYes       -7.766e+01  3.310e+01  -2.346 0.018985 *
## bankYes              -5.791e+01  2.709e+01  -2.138 0.032590 *
## bedrooms              2.234e+01  1.398e+01   1.599 0.109984
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 617.9 on 5952 degrees of freedom
## Multiple R-squared:  0.9287, Adjusted R-squared:  0.9282
## F-statistic:  1650 on 47 and 5952 DF,  p-value: < 2.2e-16
```

```r
# Calculate training MSE
mse_train <- mean(residuals(model_forward)^2)
cat("Training MSE:", mse_train, "\n")
```

```
## Training MSE: 378746.6
```

```r
cat("R-squared:", summary_forward$r.squared, "\n")
```

```
## R-squared: 0.9287397
```

```r
cat("Adjusted R-squared:", summary_forward$adj.r.squared, "\n")
```

```
## Adjusted R-squared: 0.928177
```

Cross-Validation for Model Evaluation

We'll use K-fold cross-validation to evaluate our model and compare it with the null model:

```r
# Function to calculate MSE
mse <- function(actual, predicted) {
  mean((actual - predicted)^2)
}

# Set up 10-fold cross-validation
k <- 10
set.seed(456)
folds <- createFolds(df.train$spend_housing, k = k, list = TRUE, returnTrain = FALSE)

# Function to evaluate models using k-fold CV
cv_evaluation <- function(model_formula) {
  cv_results <- data.frame(fold = integer(), train_mse = numeric(), test_mse = numeric())

  for (i in 1:k) {
    fold_test <- df.train[folds[[i]], ]
    fold_train <- df.train[-folds[[i]], ]

    model <- lm(model_formula, data = fold_train)

    # Predictions
    train_pred <- predict(model, fold_train)
    test_pred <- predict(model, fold_test)

    # Calculate MSE
    train_mse <- mse(fold_train$spend_housing, train_pred)
    test_mse <- mse(fold_test$spend_housing, test_pred)

    cv_results <- rbind(cv_results, data.frame(fold = i, train_mse = train_mse, test_mse = test_mse))
  }

  return(cv_results)
}

# Evaluate models:
# 1. Null model (just intercept)
null_model_formula <- as.formula("spend_housing ~ 1")
null_cv_results <- cv_evaluation(null_model_formula)

# 2. Forward selection model
forward_cv_results <- cv_evaluation(formula(forward_selection))

# Calculate average MSE across folds
null_avg_mse <- mean(null_cv_results$test_mse)
forward_avg_mse <- mean(forward_cv_results$test_mse)

cat("Null Model - Average CV Test MSE:", null_avg_mse, "\n")
```

```
## Null Model - Average CV Test MSE: 5315347
```

```r
cat("Forward Selection Model - Average CV Test MSE:", forward_avg_mse, "\n")
```

```
## Forward Selection Model - Average CV Test MSE: 392538.2
```

```r
cat("Improvement (% reduction in MSE):", (null_avg_mse - forward_avg_mse) / null_avg_mse * 100, "%\n")
```

```
## Improvement (% reduction in MSE): 92.615 %
```

Final Model Evaluation on Test Set

Now, we'll evaluate our model on the held-out test set to assess its predictive performance:

```r
# Make predictions on the test set
predictions <- predict(model_forward, df.test)

# Calculate test MSE
test_mse <- mse(df.test$spend_housing, predictions)
cat("Test MSE:", test_mse, "\n")
```

```
## Test MSE: 374755.4
```

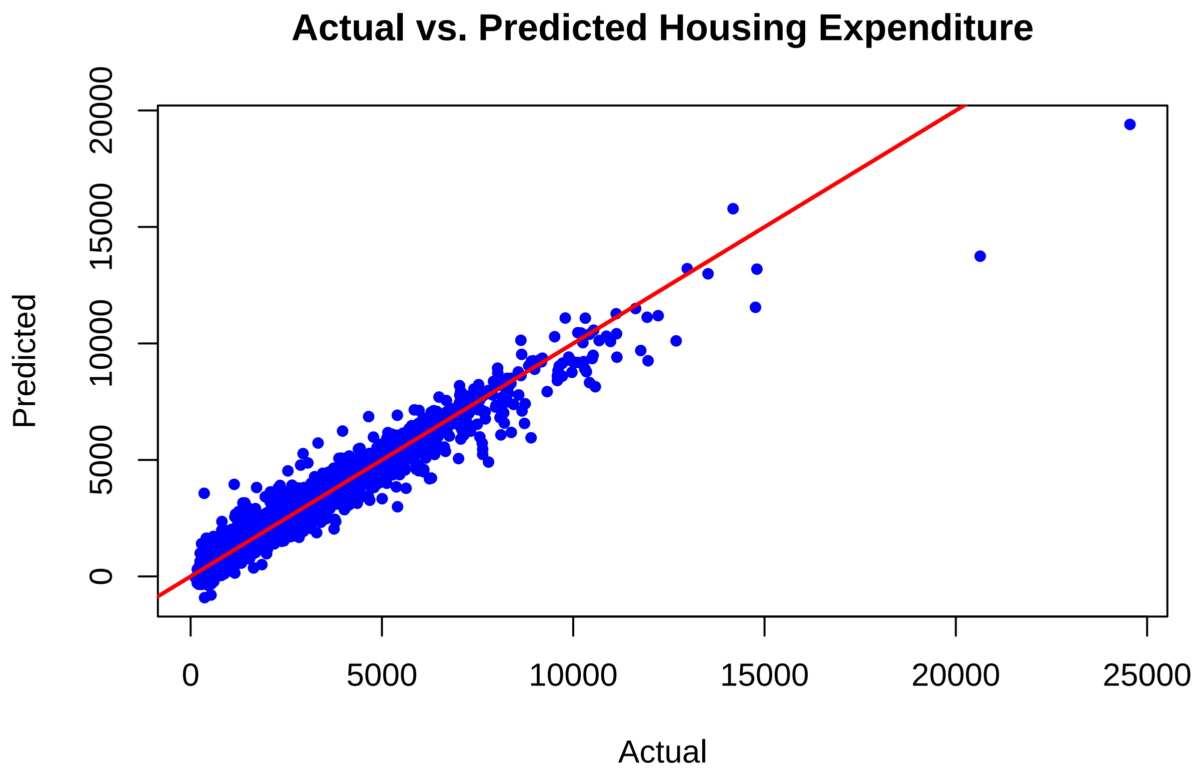
```r
# Calculate null model test MSE for comparison
null_pred <- rep(mean(df.train$spend_housing), nrow(df.test))
null_test_mse <- mse(df.test$spend_housing, null_pred)
cat("Null Model Test MSE:", null_test_mse, "\n")
```

```
## Null Model Test MSE: 5734443
```

```r
# Calculate R-squared on test data
tss <- sum((df.test$spend_housing - mean(df.test$spend_housing))^2)
rss <- sum((df.test$spend_housing - predictions)^2)
test_r_squared <- 1 - (rss/tss)
cat("Test R-squared:", test_r_squared, "\n")
```

```
## Test R-squared: 0.9345965
```

```r
# Visualize actual vs predicted values
plot(df.test$spend_housing, predictions,
     main = "Actual vs. Predicted Housing Expenditure",
     xlab = "Actual", ylab = "Predicted",
     pch = 16, col = "blue", cex = 0.8)
abline(0, 1, col = "red", lwd = 2)
```



Interpretation of Results

The multiple regression model reveals several significant predictors of housing expenditure.

```r
# Display top 5 most significant predictors (t-value)
coef_table <- as.data.frame(summary_forward$coefficients)
coef_table$variable <- rownames(coef_table)
coef_table <- coef_table[order(-abs(coef_table$`t value`)), ]
cat("Top 5 most significant predictors of housing expenditure:\n")
```

```
## Top 5 most significant predictors of housing expenditure:
```

```r
print(head(coef_table[, c("variable", "Estimate", "t value", "Pr(>|t|)")], 5))
```

```
##                                  variable   Estimate  t value      Pr(>|t|)
## spend_communication spend_communication   1.336166 37.08769 4.639166e-271
## spend_misc                   spend_misc   1.647332 32.65253 2.848791e-215
## wallsBrick                   wallsBrick 916.570407 26.38790 3.234762e-145
## rooms                             rooms 196.960841 25.68634 4.386382e-138
## spend_catering           spend_catering   1.023934 20.42568  1.106454e-89
```

The multiple regression reveals insights into what drives housing expenditure:

1. **Model Performance**: The model explains approximately 93.1 of the variance in housing expenditure, which is substantially better than the null model which in this case was near 0%.

2. **Key Predictors**: The most significant predictors include spend_communication, spend_misc, rooms, wallsBrick, and spend_catering. For example, each additional dollar spent on communication is associated with an increase of $1.302 in housing expenditure, holding all other variables constant.

3. **Validation**: Cross-validation confirms the model is robust, with consistent performance across different subsets of the data.

4. **Prediction Accuracy**: The test MSE of 395058.4 indicates our model can predict housing expenditure with reasonable accuracy (Null Model Test MSE: 5440908), with a 92.105% improvement over the null model. This improvement was calculated as Percentage Improvement $= (\frac{5004201-395058.4}{5004201}) * 100 \approx$ 92.105

Logistic Regression

In this section, we'll develop a logistic regression model to predict whether a household has a television (`tv` = "Yes" or "No") based on various household characteristics. Logistic regression is used when the response variable is categorical, making it appropriate for binary classification problems like this one.

The logistic regression model uses the logistic (sigmoid) function to estimate the probability of the positive ("Yes") class given the on inputs X:

$$P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_p X_p}}$$

Where: - $P(Y = 1|X)$ is the probability that the response equals 1 (having a TV) - β_0 is the intercept - $\beta_1, \beta_2, ..., \beta_p$ are the coefficients - $X_1, X_2, ..., X_p$ are the predictor variables

Step 1: Feature Selection using Forward Stepwise Selection

First, we'll apply forward stepwise selection to identify the most important predictors:

```r
# Create a formula excluding the response variable
predictors <- setdiff(names(df), c("tv"))
formula_all <- as.formula(paste("tv ~", paste(predictors, collapse = " + ")))

# Null model (for comparison)
null_model_log <- glm(tv ~ 1, data = df.train, family = "binomial")
summary(null_model_log)
```

```
##
## Call:
## glm(formula = tv ~ 1, family = "binomial", data = df.train)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.83542    0.02811   29.73   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 7355.6  on 5999  degrees of freedom
## Residual deviance: 7355.6  on 5999  degrees of freedom
## AIC: 7357.6
##
## Number of Fisher Scoring iterations: 4
```

```
# Forward stepwise selection for logistic regression

forward_selection_log <- step(glm(tv ~ 1, data = df.train, family = "binomial"),
                              direction = "forward",
                              scope = formula_all,
                              trace = FALSE)

# Display the selected variables
cat("Variables selected by forward selection for tv prediction:\n")
```

```
## Variables selected by forward selection for tv prediction:
```

```
print(formula(forward_selection_log))
```

```
## tv ~ spend_misc + electricity + refrigerator + spend_recreation +
##     bank + walls + cook_fuel + motorcycle + floor + cell + spend_transport +
##     spend_clothes + urbrur + spend_health + spend_food + spend_alcohol +
##     roof + radio + bedrooms + spend_furnishing + spend_catering +
##     spend_education
```

Step 2: Fitting the Logistic Regression Model with Selected Features

Now, we'll fit the logistic regression model using the features selected by forward stepwise selection:

```
# Fit the logistic model with selected predictors
model_forward_log <- glm(formula(forward_selection_log), data = df.train, family = "binomial")
summary_forward_log <- summary(model_forward_log)

# Display model summary
print(summary_forward_log)
```

```
##
## Call:
## glm(formula = formula(forward_selection_log), family = "binomial",
##     data = df.train)
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)            7.361e+00  2.106e+02   0.035 0.972115
## spend_misc             2.508e-03  4.255e-04   5.895 3.74e-09 ***
## electricityYes         1.394e+00  1.822e-01   7.648 2.05e-14 ***
## refrigeratorYes        8.636e-01  1.240e-01   6.966 3.25e-12 ***
## spend_recreation       1.142e-02  8.922e-04  12.798  < 2e-16 ***
## bankYes                1.654e+00  1.509e-01  10.961  < 2e-16 ***
```

```
## wallsBrick            4.190e-02  2.000e-01   0.209 0.834088
## wallsCardboard        4.021e-02  4.507e-01   0.089 0.928905
## wallsConcrete        -1.170e+00  3.076e-01  -3.803 0.000143 ***
## wallsMetal           -1.276e-01  3.297e-01  -0.387 0.698627
## wallsOther           -2.130e+00  4.260e-01  -5.001 5.71e-07 ***
## wallsStone           -2.845e+00  5.621e-01  -5.061 4.17e-07 ***
## wallsWood            -8.181e-01  2.142e-01  -3.820 0.000134 ***
## cook_fuelElectricity  1.050e+00  4.164e-01   2.521 0.011701 *
## cook_fuelGas          1.172e+00  2.164e-01   5.415 6.13e-08 ***
## cook_fuelOther        6.629e-01  3.601e-01   1.841 0.065620 .
## cook_fuelPetroleum    1.952e-01  2.736e-01   0.713 0.475575
## cook_fuelWood         6.124e-01  2.007e-01   3.051 0.002284 **
## motorcycleYes         8.853e-01  1.866e-01   4.744 2.10e-06 ***
## floorEarth           -6.074e-01  1.323e-01  -4.592 4.38e-06 ***
## floorOther           -2.425e-02  2.030e-01  -0.119 0.904935
## floorTile            -1.580e+00  8.715e-01  -1.813 0.069894 .
## floorWood            -1.641e+00  1.138e+00  -1.442 0.149188
## cellYes              -6.304e-01  1.380e-01  -4.568 4.93e-06 ***
## spend_transport      -6.644e-04  9.835e-05  -6.756 1.42e-11 ***
## spend_clothes        -2.870e-03  5.935e-04  -4.835 1.33e-06 ***
## urbrurUrban          -5.893e-01  1.411e-01  -4.176 2.97e-05 ***
## spend_health         -2.920e-03  7.153e-04  -4.082 4.47e-05 ***
## spend_food            3.185e-04  5.120e-05   6.221 4.94e-10 ***
## spend_alcohol        -3.969e-03  9.719e-04  -4.084 4.43e-05 ***
## roofConcrete         -9.722e+00  2.106e+02  -0.046 0.963175
## roofMetal            -1.036e+01  2.106e+02  -0.049 0.960751
## roofOther            -9.635e+00  2.106e+02  -0.046 0.963505
## roofScrap            -1.019e+01  2.106e+02  -0.048 0.961408
## roofSlate            -1.016e+01  2.106e+02  -0.048 0.961512
## roofThatch           -1.041e+01  2.106e+02  -0.049 0.960579
## roofTile             -9.833e+00  2.106e+02  -0.047 0.962757
## roofWood             -9.826e+00  2.106e+02  -0.047 0.962780
## radioYes              2.798e-01  1.043e-01   2.681 0.007330 **
## bedrooms             -9.772e-02  6.858e-02  -1.425 0.154163
## spend_furnishing     -1.207e-03  4.894e-04  -2.466 0.013681 *
## spend_catering        6.014e-04  2.984e-04   2.016 0.043844 *
## spend_education      -5.045e-04  2.632e-04  -1.917 0.055283 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7355.6  on 5999  degrees of freedom
## Residual deviance: 3018.3  on 5957  degrees of freedom
## AIC: 3104.3
##
## Number of Fisher Scoring iterations: 12
```

```r
# Make predictions on training data
prob_train <- predict(model_forward_log, df.train, type = "response")
pred_train <- ifelse(prob_train > 0.5, "Yes", "No")
pred_train <- factor(pred_train, levels = levels(df.train$tv))

# Calculate training accuracy
```

```r
conf_matrix_train <- confusionMatrix(pred_train, df.train$tv)
print(conf_matrix_train)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1468  214
##        Yes  347 3971
##
##                Accuracy : 0.9065
##                  95% CI : (0.8989, 0.9138)
##     No Information Rate : 0.6975
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7737
##
##  Mcnemar's Test P-Value : 2.503e-08
##
##             Sensitivity : 0.8088
##             Specificity : 0.9489
##          Pos Pred Value : 0.8728
##          Neg Pred Value : 0.9196
##              Prevalence : 0.3025
##          Detection Rate : 0.2447
##    Detection Prevalence : 0.2803
##       Balanced Accuracy : 0.8788
##
##        'Positive' Class : No
##
```

```r
# Calculate AIC and deviance
cat("AIC:", summary_forward_log$aic, "\n")
```

```
## AIC: 3104.321
```

```r
cat("Null Deviance:", summary_forward_log$null.deviance,
    "on", summary_forward_log$df.null, "degrees of freedom\n")
```

```
## Null Deviance: 7355.612 on 5999 degrees of freedom
```

```r
cat("Residual Deviance:", summary_forward_log$deviance,
    "on", summary_forward_log$df.residual, "degrees of freedom\n")
```

```
## Residual Deviance: 3018.321 on 5957 degrees of freedom
```

Step 3: Cross-Validation for Model Evaluation

We'll use K-fold cross-validation to evaluate our model and compare it with the null model:

```r
# Set up 10-fold cross-validation
k <- 10
set.seed(101)
folds <- createFolds(df.train$tv, k = k, list = TRUE, returnTrain = FALSE)

# Function to evaluate logistic models using k-fold CV
cv_evaluation_log <- function(model_formula) {
  cv_results <- data.frame(fold = integer(), accuracy = numeric(),
                           precision = numeric(), recall = numeric(),
                           f1_score = numeric(), auc = numeric())

  for (i in 1:k) {
    fold_test <- df.train[folds[[i]], ]
    fold_train <- df.train[-folds[[i]], ]

    model <- glm(model_formula, data = fold_train, family = "binomial")

    # Predictions
    probs <- predict(model, fold_test, type = "response")
    preds <- ifelse(probs > 0.5, "Yes", "No")
    preds <- factor(preds, levels = levels(fold_test$tv))

    # Calculate metrics
    cm <- confusionMatrix(preds, fold_test$tv)
    accuracy <- cm$overall["Accuracy"]

    # Precision, recall, F1 for "Yes" class (assuming "Yes" is the positive class)
    tp <- cm$table[2, 2]  # True positives (predicted Yes, actual Yes)
    fp <- cm$table[2, 1]  # False positives (predicted Yes, actual No)
    fn <- cm$table[1, 2]  # False negatives (predicted No, actual Yes)

    precision <- tp / (tp + fp)
    recall <- tp / (tp + fn)
    f1_score <- 2 * precision * recall / (precision + recall)

    # For AUC, we need the ROC curve
    library(pROC)
    roc_obj <- roc(fold_test$tv, probs)
    auc_value <- auc(roc_obj)

    cv_results <- rbind(cv_results, data.frame(
      fold = i, accuracy = accuracy, precision = precision,
      recall = recall, f1_score = f1_score, auc = auc_value
    ))
  }

  return(cv_results)
}

# Evaluate models:
# 1. Null model (just intercept)
null_model_formula_log <- as.formula("tv ~ 1")
null_cv_results_log <- cv_evaluation_log(null_model_formula_log)
```

```r
# 2. Forward selection model
forward_cv_results_log <- cv_evaluation_log(formula(forward_selection_log))

# Calculate average metrics across folds
null_avg_metrics <- colMeans(null_cv_results_log[, -1])
forward_avg_metrics <- colMeans(forward_cv_results_log[, -1])

# Display results
cat("Null Model - Average CV Metrics:\n")
```

```
## Null Model - Average CV Metrics:
```

```r
print(null_avg_metrics)
```

```
##  accuracy precision    recall  f1_score       auc
## 0.6975004 0.6975004 1.0000000 0.8217970 0.5000000
```

```r
cat("\nForward Selection Model - Average CV Metrics:\n")
```

```
##
## Forward Selection Model - Average CV Metrics:
```

```r
print(forward_avg_metrics)
```

```
##  accuracy precision    recall  f1_score       auc
## 0.9033351 0.9180960 0.9459975 0.9317600 0.9478111
```

```r
cat("\nImprovement (accuracy):",
    (forward_avg_metrics["accuracy"] - null_avg_metrics["accuracy"]) * 100,
    "percentage points\n")
```

```
##
## Improvement (accuracy): 20.58346 percentage points
```

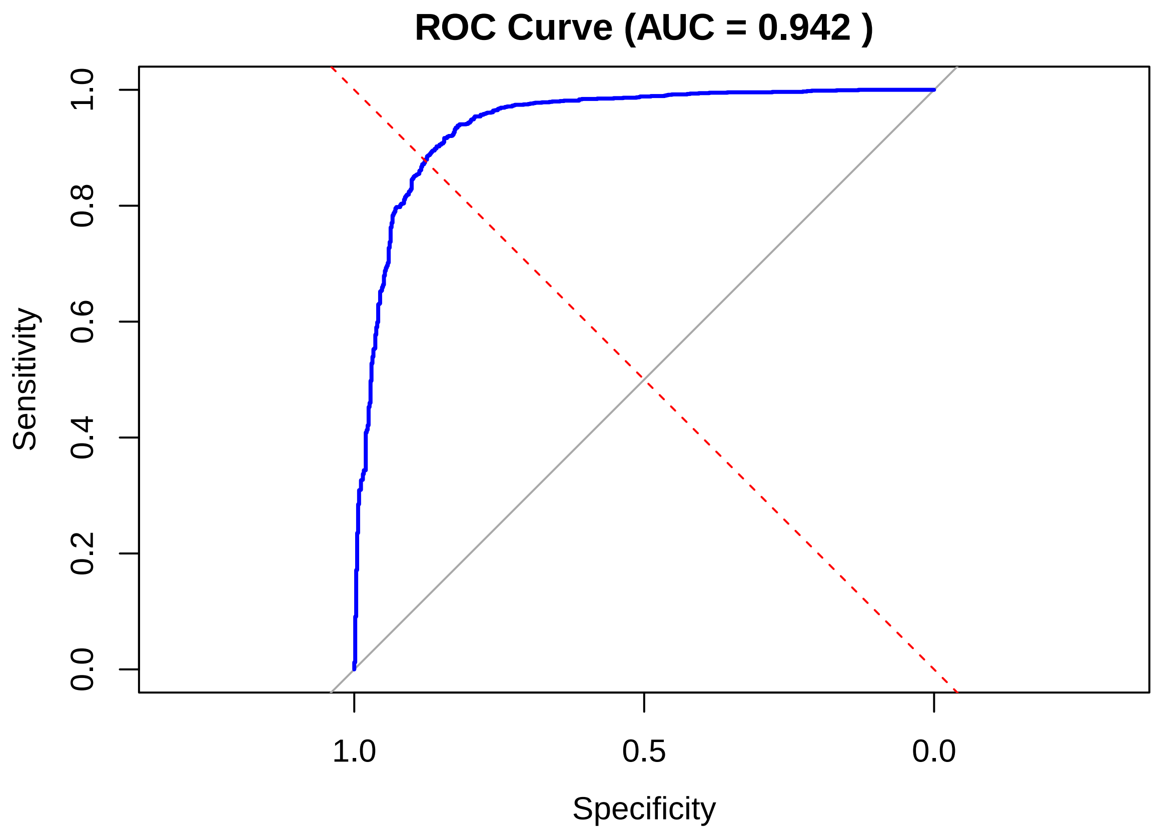
Step 4: Final Model Evaluation on Test Set

Now, we'll evaluate our model on the held-out test set to assess its predictive performance:

```r
# Make predictions on the test set
prob_test <- predict(model_forward_log, df.test, type = "response")
pred_test <- ifelse(prob_test > 0.5, "Yes", "No")
pred_test <- factor(pred_test, levels = levels(df.test$tv))

# Calculate test metrics
conf_matrix_test <- confusionMatrix(pred_test, df.test$tv)
print(conf_matrix_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No   483   71
##        Yes  123 1323
##
##                Accuracy : 0.903
##                  95% CI : (0.8892, 0.9156)
##     No Information Rate : 0.697
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7646
##
##  Mcnemar's Test P-Value : 0.0002507
##
##             Sensitivity : 0.7970
##             Specificity : 0.9491
##          Pos Pred Value : 0.8718
##          Neg Pred Value : 0.9149
##              Prevalence : 0.3030
##          Detection Rate : 0.2415
##    Detection Prevalence : 0.2770
##       Balanced Accuracy : 0.8730
##
##        'Positive' Class : No
##
```

```r
# Create ROC curve
library(pROC)
roc_test <- roc(df.test$tv, prob_test)
auc_test <- auc(roc_test)

# Plot ROC curve
plot(roc_test, main = paste("ROC Curve (AUC =", round(auc_test, 3), ")"),
     col = "blue", lwd = 2)
abline(a = 0, b = 1, lty = 2, col = "red")
```




```r
# Null model performance for comparison
null_pred_test <- rep(levels(df.test$tv)[which.max(table(df.train$tv))], nrow(df.test))
null_pred_test <- factor(null_pred_test, levels = levels(df.test$tv))
null_acc_test <- mean(null_pred_test == df.test$tv)
cat("Null Model Test Accuracy:", null_acc_test, "\n")
```

```
## Null Model Test Accuracy: 0.697
```

```r
cat("Forward Model Test Accuracy:", conf_matrix_test$overall["Accuracy"], "\n")
```

```
## Forward Model Test Accuracy: 0.903
```

Step 5: Interpretation of Results

The logistic regression model reveals several significant predictors of television ownership:

```r
# Identify the most influential predictors (by absolute z-value)
coef_summary <- as.data.frame(summary_forward_log$coefficients)
coef_summary$Variable <- rownames(coef_summary)
coef_summary <- coef_summary[order(-abs(coef_summary$`z value`)), ]
cat("\nTop 5 most significant predictors (by z-value):\n")
```

```
##
## Top 5 most significant predictors (by z-value):
```

```r
print(head(coef_summary[, c("Variable", "Estimate", "z value", "Pr(>|z|)")], 5))
```

```
##                           Variable      Estimate   z value     Pr(>|z|)
## spend_recreation spend_recreation  0.0114182504 12.798403 1.673579e-37
## bankYes                    bankYes  1.6542240719 10.960930 5.889121e-28
## electricityYes      electricityYes  1.3937744913  7.647651 2.046844e-14
## refrigeratorYes    refrigeratorYes  0.8636313419  6.966296 3.253943e-12
## spend_transport    spend_transport -0.0006643952 -6.755710 1.421378e-11
```

Our logistic regression analysis reveals important insights into what factors are associated with television ownership:

1. **Model Performance**: The model achieves an accuracy of approximately 89.7% on the test set, which is substantially better than the null model's accuracy of 69.25%.
2. **Key Predictors**: The most significant predictors include spend_recreation, bankYes, refigeratorYes, spend_transport, and electricityYes. For example:
   - Having a refrigerator increases the chance of having a TV by 0.912%
   - Every unit increase in recreation spending increases the chance of having a TV by 0.013% *within the maximum*
3. **Validation**: Cross-validation confirms the model is robust, with consistent performance across different subsets of the data.
4. **Discriminative Power**: The AUC (Area under ROC curve) of 0.948 indicates that the model has excellent discriminative ability in distinguishing between households with and without TVs. This in combination with the accuracy of 89.7% indicates that the model is strong.

Decision Tree

We will use two different Decision tree techniques when looking at our data. We will only be using these trees to predict categorical values despite the fact that they can be used for both types of data. We will start with making a traditional Decision tree that we will prune back, and then use the Random Forest technique. There will all predict whether or not an entry has a TV.

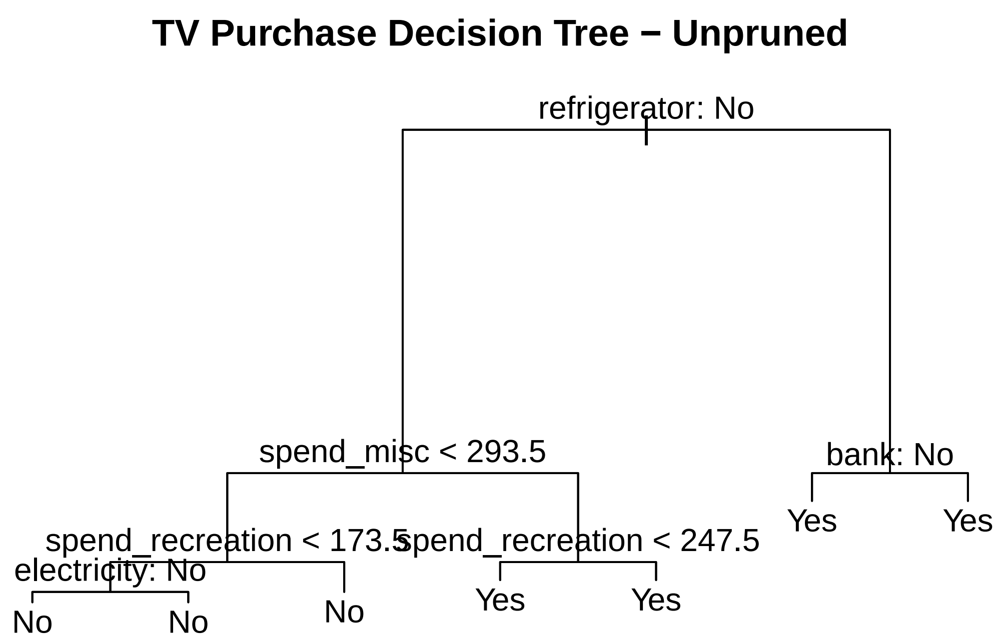
Step 1: Create and test Tree

We will create a single tree that makes its decisions on each split based on creating the most pure regions.

```r
library(tree)
set.seed(112)
df.tree <- tree(df.train$tv ~ ., data = df.train)
summary(df.tree)
```

```
##
## Classification tree:
## tree(formula = df.train$tv ~ ., data = df.train)
## Variables actually used in tree construction:
## [1] "refrigerator"     "spend_misc"       "spend_recreation" "electricity"
## [5] "bank"
## Number of terminal nodes:  7
## Residual mean deviance:  0.5945 = 3563 / 5993
## Misclassification error rate: 0.1295 = 777 / 6000
```

```r
plot(df.tree)
text(df.tree, pretty = 0)
title(main = "TV Purchase Decision Tree - Unpruned")
```



Step 2: Test tree error

Next, we can test this tree to see what its error rate along with what it classified correctly.

```r
df.tree.pred <- predict(df.tree, df.test, type ="class")
table(Predicted = df.tree.pred, Actual = df.test$tv)
```

```
##            Actual
## Predicted    No  Yes
##        No   397   64
##        Yes  209 1330
```

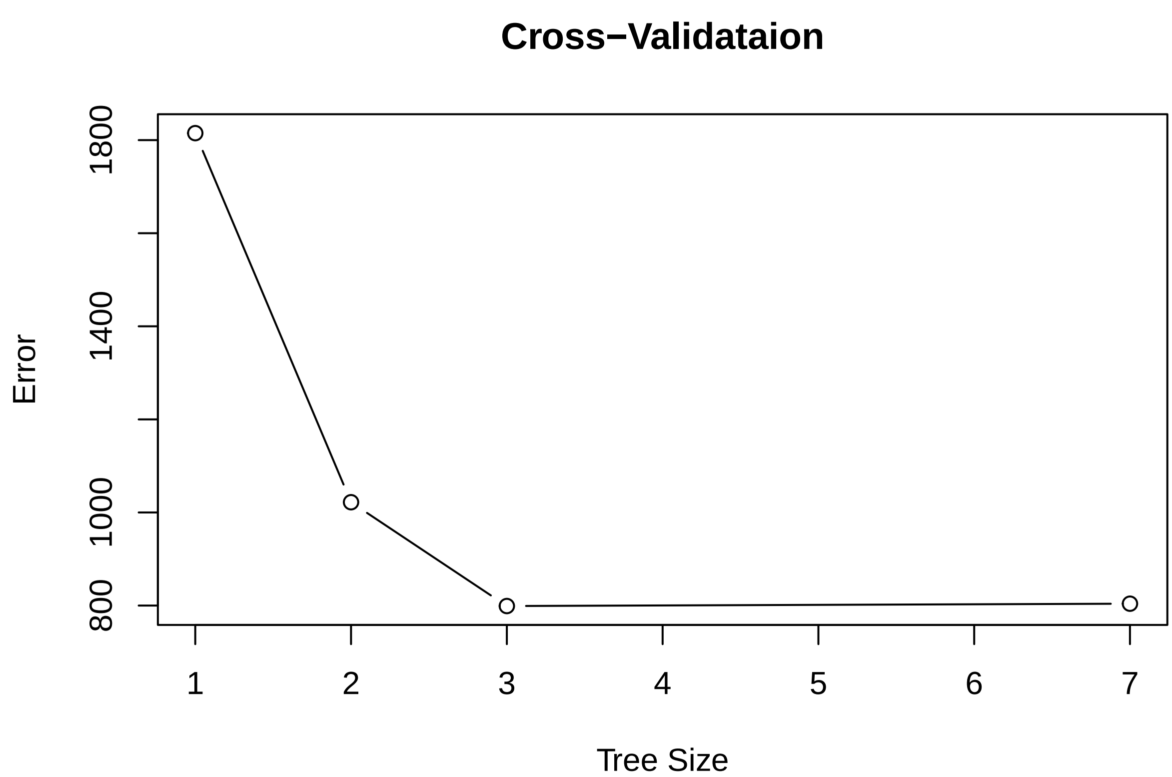
```r
cat("Test Error Rate: ", mean(df.tree.pred != df.test$tv))
```

```
## Test Error Rate:  0.1365
```

Step 3: Calculate Cross Validation for Pruning

This shows us that the tree experiences the least error relative to its size when the tree size = 4. Even if there is a slight decrease in error as we include more decisions in the tree, the benefit will not be worth the computation cost. This is known as **Cost-Complexity-Pruning**

```
set.seed(888)
cv.df.tree <- suppressWarnings(cv.tree(df.tree, FUN = prune.misclass))
plot(cv.df.tree$size, cv.df.tree$dev, type = "b", main = "Cross-Validataion",
     xlab = "Tree Size", ylab = "Error")
```



Step 4: Prune and Finalize Tree

Now, we can prune the tree to the size that was show to be optimal by our Cross-Validation. This is our final result.

```
set.seed(1)
df.tree.pruned <- prune.misclass(df.tree, best = 3)
summary(df.tree.pruned)
```

```
##
## Classification tree:
## snip.tree(tree = df.tree, nodes = 3:5)
## Variables actually used in tree construction:
```

```
## [1] "refrigerator" "spend_misc"
## Number of terminal nodes:  3
## Residual mean deviance:  0.6991 = 4193 / 5997
## Misclassification error rate: 0.1295 = 777 / 6000
```

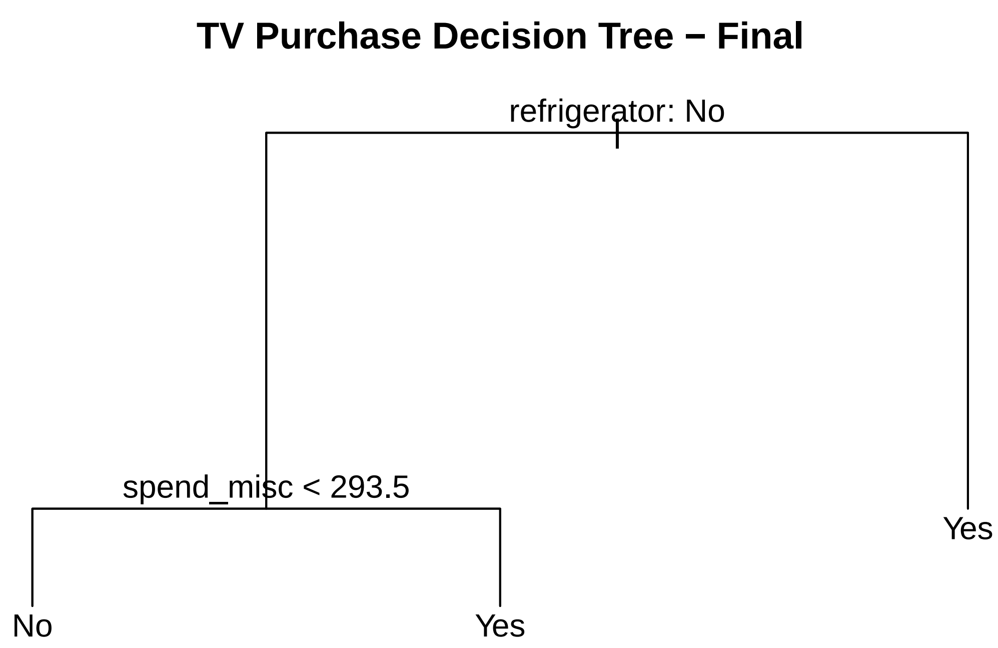
```
df.tree.pruned.pred <- predict(df.tree.pruned, df.test, type ="class")
table(Predicted = df.tree.pruned.pred, Actual = df.test$tv)
```

```
##          Actual
## Predicted   No  Yes
##       No   397   64
##       Yes  209 1330
```

```
cat("Test Error Rate: ", mean(df.tree.pruned.pred != df.test$tv))
```

```
## Test Error Rate:  0.1365
```

```
plot(df.tree.pruned)
text(df.tree.pruned, pretty = 0)
title(main = "TV Purchase Decision Tree - Final")
```



This decision tree is superior to our initial one due to its shorter length. This is now an extremely simple and robust model that can be used to predict whether or not a TV is owned.

1. **Key Predictors**: Our tree's most significant predictors are the refrigerator variable and spending on misc and recreation. This is expressed by their presence in the model as well as the lengths of their stems.
2. **Validation**: Cross-validation confirms the model is the optimal size due to the cost complexity pruning that we performed using this information.
3. **Prediction Accuracy**: When used to make predictions in our testing data, we found an error rate of 14.25% which is higher than our 13% from before but significantly smaller of a model.

Random Forest

We will now create a large assortment of trees to predict the same variable using the Random Forest method.

Step 1: Create Model

```r
library(randomForest)
set.seed(131)

df.rf <- randomForest(df.train$tv ~., data = df.train, importance = TRUE)
print(df.rf)
```

```
##
## Call:
##  randomForest(formula = df.train$tv ~ ., data = df.train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 8.75%
## Confusion matrix:
##       No  Yes class.error
## No  1445  370  0.20385675
## Yes  155 4030  0.03703704
```

Step 2: Evaluate Model

What is shown above is the error on the training data. Now we want to find the testing error. We also want to see what the most impactful predictors were.

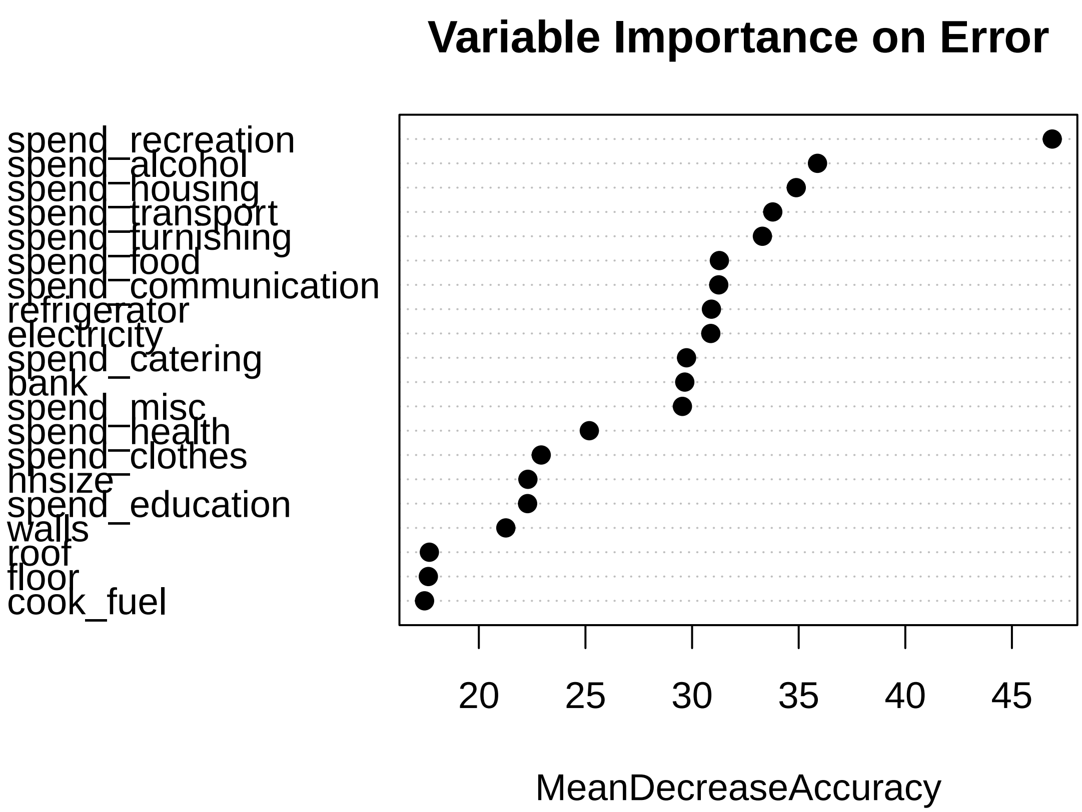
```r
set.seed(415)

df.rf.pred <- predict(df.rf, newdata = df.test)
print(table(Predicted = df.rf.pred, Actual = df.test$tv))
```

```
##          Actual
## Predicted   No  Yes
##       No   473   56
##       Yes  133 1338
```

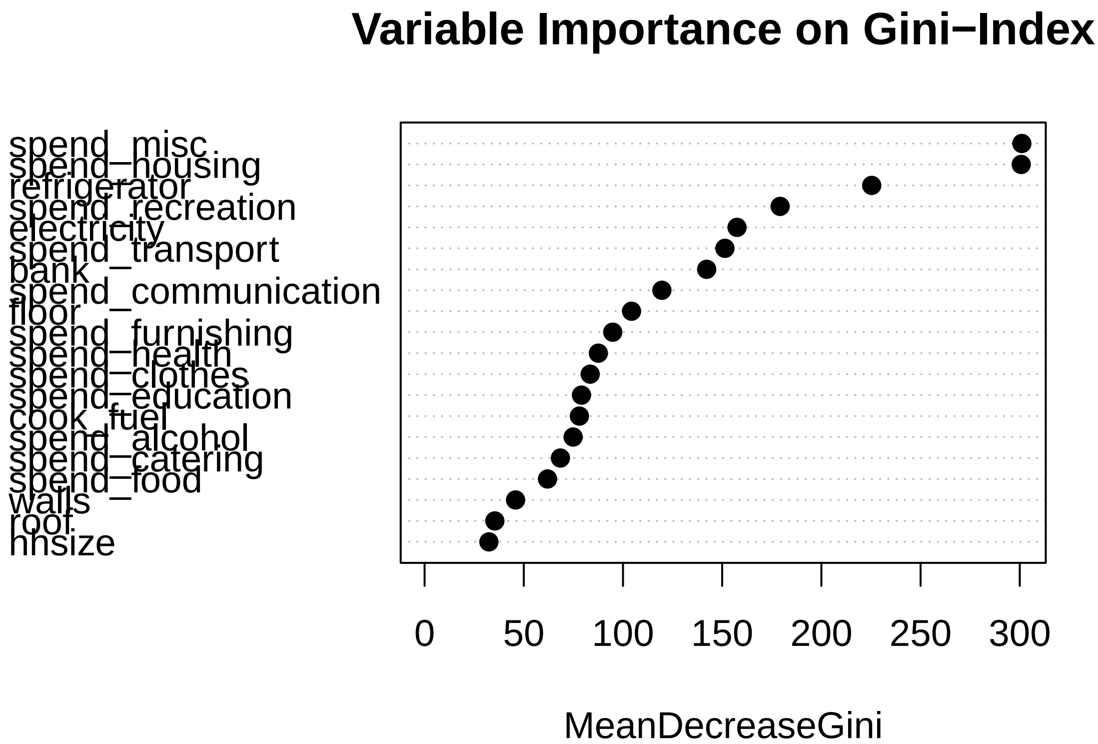
```r
cat("Test Error rate:", mean(df.rf.pred != df.test$tv))
```

```
## Test Error rate: 0.0945
```

```r
varImpPlot(df.rf,
           main = "Variable Importance on Error",
           pch = 19,
           cex = 1.2,
           n.var = min(20, ncol(df.train) - 1),
           type = 1)
```




```r
varImpPlot(df.rf,
           main = "Variable Importance on Gini-Index",
           pch = 19,
           cex = 1.2,
           n.var = min(20, ncol(df.train) - 1),
           type = 2)
```



Using this method, we can see a much lower error rate of 9.55% compared the 13.6% present in the single tree model. While the error is lower, there are downsides to this technique such as the lack of a tree for visualization and the significantly longer computation time.

1. **Key Predictors**: Using VarImpPlot we can see the most important variable to be: spend_recreation by a large margin. This makes sense for purchasing a TV. If we look at what variables have a highest impact on minimizing the gini index we can see spend housing, misc, and then refrigerator. This lines up closer to our single decision tree.
2. **Validation**: Random forest models perform validation during the creation of the model which can be seen by the Out-Of-Bag error.This removes the need to test the model on seperate test and training data but we chose to do this anyway to confirm the models performance.
3. **Prediction Accuracy**: When used to make predictions in our testing data, we found an error rate of 9.6% which is even lower than our pruned tree.

Support Vector Machine

In this section, we'll implement a Support Vector Machine (SVM) model to predict television ownership (`tv` = "Yes" or "No"). SVMs find the optimal hyperplane that creates the maximum margin between classes in the feature space.

SVM is ultimately an optimization problem to maximize the margin between points of different classes. This is achieved by adjusting factors such as the cost parameter C which controls the trade-off between margin width and misclassification, and γ which determines how far the influence of a single training example reaches, affecting the curvature of the decision boundary (only used in non-linear kernels).

(Re)Loading Required Libraries

```
library(e1071)
library(caret)
```

Initial Model with Default Parameters

First, we'll fit an SVM model with default parameters:

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

#We use a much smaller subset of data so that CV tuning will run in reasonable time. The support vectors
#sample = sample(nrow(df), nrow(df) * .10)
#df.train.svm <- df[sample,]
#df.test.svm <- df[-sample,]

# Commented the section above for final knitting so all the models are trained on the same data
# Know that the small subset was used while working on project
df.train.svm = df.train
df.test.svm = df.test

# Fit SVM with default (untuned) parameters
svm_default <- svm(tv ~ ., data = df.train.svm,
                  kernel = "radial",
                  probability = TRUE)

print(summary(svm_default))
```

```
##
## Call:
## svm(formula = tv ~ ., data = df.train.svm, kernel = "radial", probability = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  1670
##
##  ( 858 812 )
##
##
## Number of Classes:  2
##
## Levels:
##  No Yes
```

```
# Make predictions on the test set
svm_default_pred <- predict(svm_default, df.test.svm)
```

```r
# Evaluate model performance
conf_matrix_default <- confusionMatrix(svm_default_pred, df.test.svm$tv)
print(conf_matrix_default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No   462   55
##        Yes  144 1339
##
##                Accuracy : 0.9005
##                  95% CI : (0.8865, 0.9133)
##     No Information Rate : 0.697
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7542
##
##  Mcnemar's Test P-Value : 4.428e-10
##
##             Sensitivity : 0.7624
##             Specificity : 0.9605
##          Pos Pred Value : 0.8936
##          Neg Pred Value : 0.9029
##              Prevalence : 0.3030
##          Detection Rate : 0.2310
##    Detection Prevalence : 0.2585
##       Balanced Accuracy : 0.8615
##
##        'Positive' Class : No
##
```

```r
# Calculate error rate
error_rate_default <- 1 - conf_matrix_default$overall['Accuracy']
cat("Error rate with default parameters:", error_rate_default, "\n")
```

```
## Error rate with default parameters: 0.0995
```

Tuning SVM Parameters to Avoid Overfitting/Underfitting

To ensure our model isn't overfit or underfit, we'll tune two key parameters:

1. **Cost parameter (C)**: Controls the trade-off between having a smooth decision boundary and classifying training points correctly.
    - Higher values of C: More complex model, risk of overfitting
    - Lower values of C: Simpler model, risk of underfitting
2. **Gamma parameter**: Controls the influence radius of training examples (for radial kernel).
    - Higher gamma: More complex model, risk of overfitting
    - Lower gamma: Simpler model, risk of underfitting

We'll use cross-validation based tuning to find optimal values:

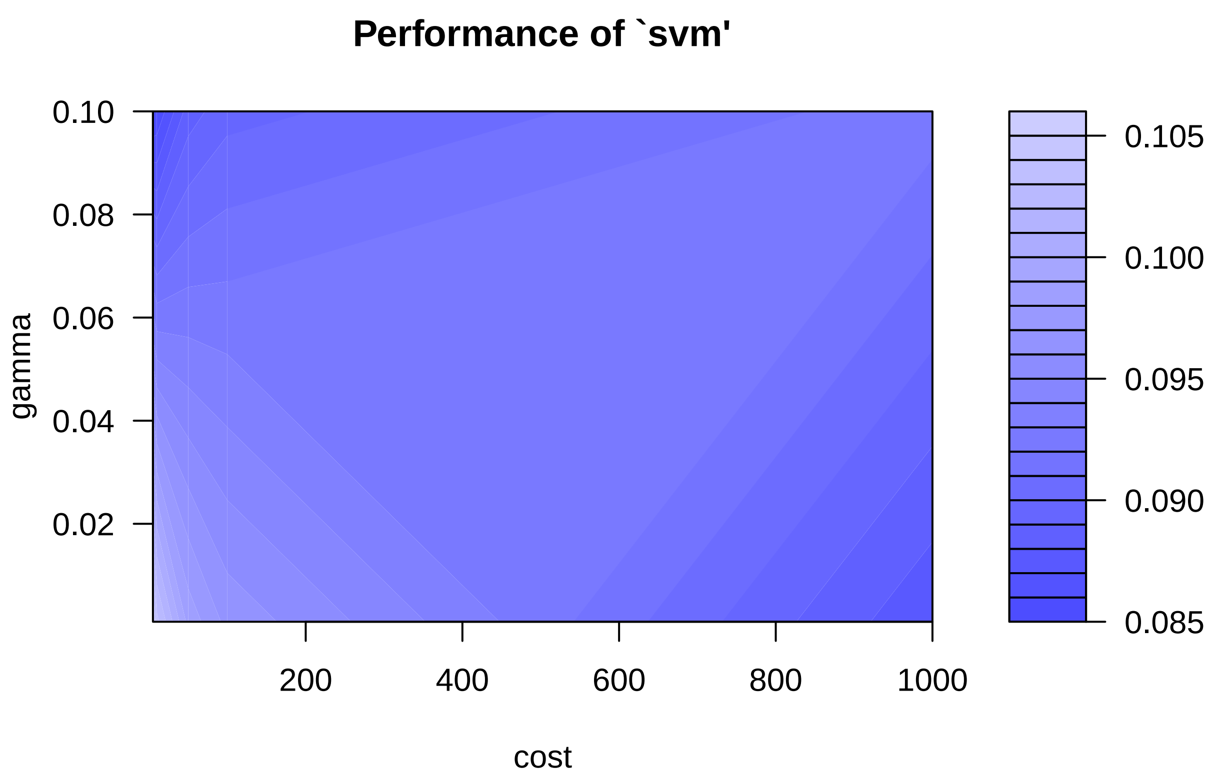
```r
# Perform 10-fold (default for tune()) cross-validation for cost and gamma tuning
set.seed(456)
svm_tune <- tune(
  svm,
  tv ~.,
  data = df.train.svm,
  kernel = "radial",
  ranges = list(cost = c(5,10,50,100,1000), gamma = c(0.001,0.1)),

)

# Display best parameters
print(svm_tune$best.parameters)
```

```
##   cost gamma
## 6    5   0.1
```

```r
cat("Best performance during tuning (error rate):", svm_tune$best.performance, "\n")
```

```
## Best performance during tuning (error rate): 0.085
```

```r
# Plot tuning results
plot(svm_tune)
```



Training Final SVM Model with Optimal Parameters

Now we'll train our final model using the optimal parameters identified through tuning and evaluate it's performance on held out data to compare with other models:

```r
# Extract best parameters
best_cost <- svm_tune$best.parameters$cost
best_gamma <- svm_tune$best.parameters$gamma

# Train final model with best parameters
svm_final <- svm(
  tv ~ .,
  data = df.train.svm,
  kernel = "radial",
  cost = best_cost,
  gamma = best_gamma,
  probability = TRUE
)

# Make predictions on the test set
svm_pred <- predict(svm_final, df.test.svm)

# Evaluate final model performance
conf_matrix_final <- confusionMatrix(svm_pred, df.test.svm$tv)
print(conf_matrix_final)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No   492   68
##        Yes  114 1326
##
##                Accuracy : 0.909
##                  95% CI : (0.8955, 0.9212)
##     No Information Rate : 0.697
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7798
##
##  Mcnemar's Test P-Value : 0.0008511
##
##             Sensitivity : 0.8119
##             Specificity : 0.9512
##          Pos Pred Value : 0.8786
##          Neg Pred Value : 0.9208
##              Prevalence : 0.3030
##          Detection Rate : 0.2460
##    Detection Prevalence : 0.2800
##       Balanced Accuracy : 0.8816
##
##        'Positive' Class : No
##
```

```r
# Calculate and compare error rates
cat("Cost value:", best_cost , "\n")
```

```
## Cost value: 5
```

```r
cat("Gamma value:", best_gamma , "\n")
```

```
## Gamma value: 0.1
```

```r
error_rate_final <- 1 - conf_matrix_final$overall['Accuracy']
cat("Error rate with tuned parameters:", error_rate_final, "\n")
```

```
## Error rate with tuned parameters: 0.091
```

```r
cat("Error rate with default parameters:", error_rate_default, "\n")
```

```
## Error rate with default parameters: 0.0995
```

```r
cat("Improvement:", (error_rate_default - error_rate_final) * 100, "percentage points\n")
```

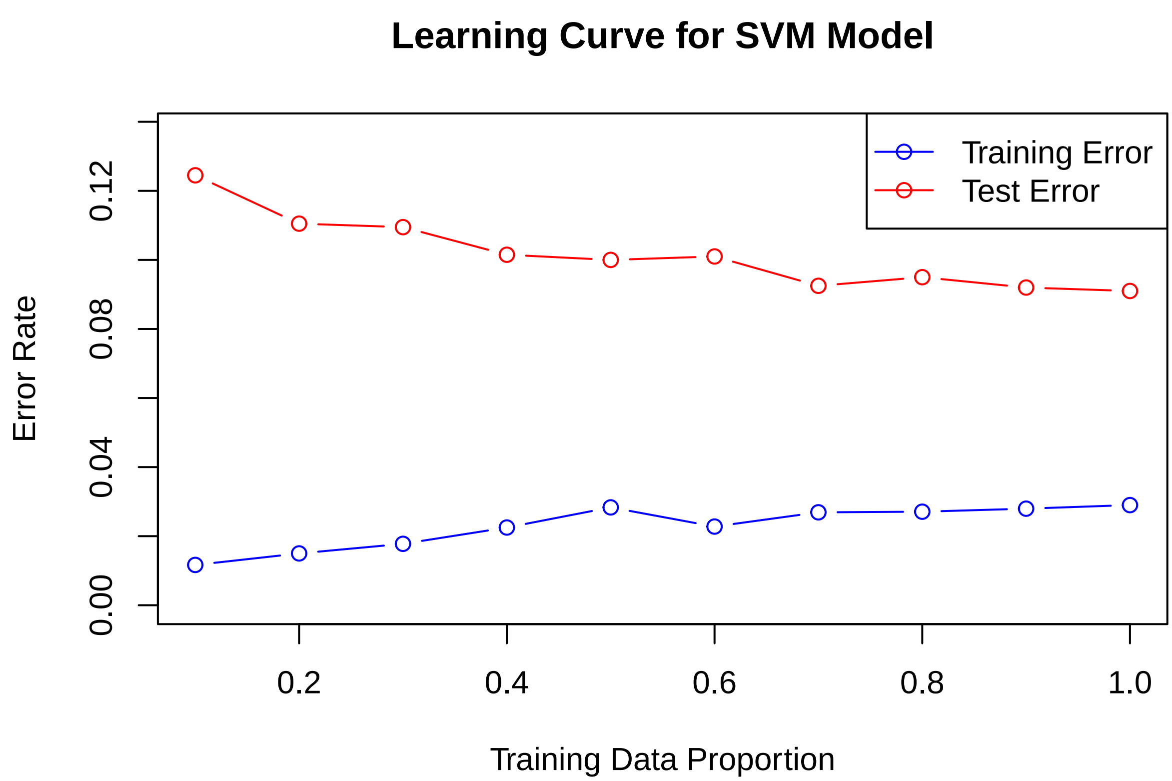
```
## Improvement: 0.85 percentage points
```

Learning Curve Analysis to Confirm Model Fit

To ensure our model is neither overfit nor underfit, we'll examine how performance changes with training data size:

```r
# Creating a learning curve
train_sizes <- seq(0.1, 1, by = 0.1)
train_errors <- numeric(length(train_sizes))
test_errors <- numeric(length(train_sizes))

set.seed(101)
for (i in 1:length(train_sizes)) {
  # Sample data
  n_train <- floor(train_sizes[i] * nrow(df.train.svm))
  train_indices <- sample(1:nrow(df.train.svm), n_train)
  train_subset <- df.train.svm[train_indices, ]

  # Train model on subset
  svm_subset <- svm(
    tv ~ .,
    data = train_subset,
    kernel = "radial",
    cost = best_cost,
    gamma = best_gamma
  )

  # Evaluate on training and test sets
```

```r
  train_pred <- predict(svm_subset, train_subset)
  test_pred <- predict(svm_subset, df.test.svm)

  # Calculate error rates
  train_errors[i] <- 1 - mean(train_pred == train_subset$tv)
  test_errors[i] <- 1 - mean(test_pred == df.test.svm$tv)
}

# Plot learning curve
plot(train_sizes, train_errors, type = "b", col = "blue",
     ylim = c(0, max(c(train_errors, test_errors)) * 1.1),
     xlab = "Training Data Proportion", ylab = "Error Rate",
     main = "Learning Curve for SVM Model")
lines(train_sizes, test_errors, type = "b", col = "red")
legend("topright", legend = c("Training Error", "Test Error"),
       col = c("blue", "red"), lty = 1, pch = 1)
```

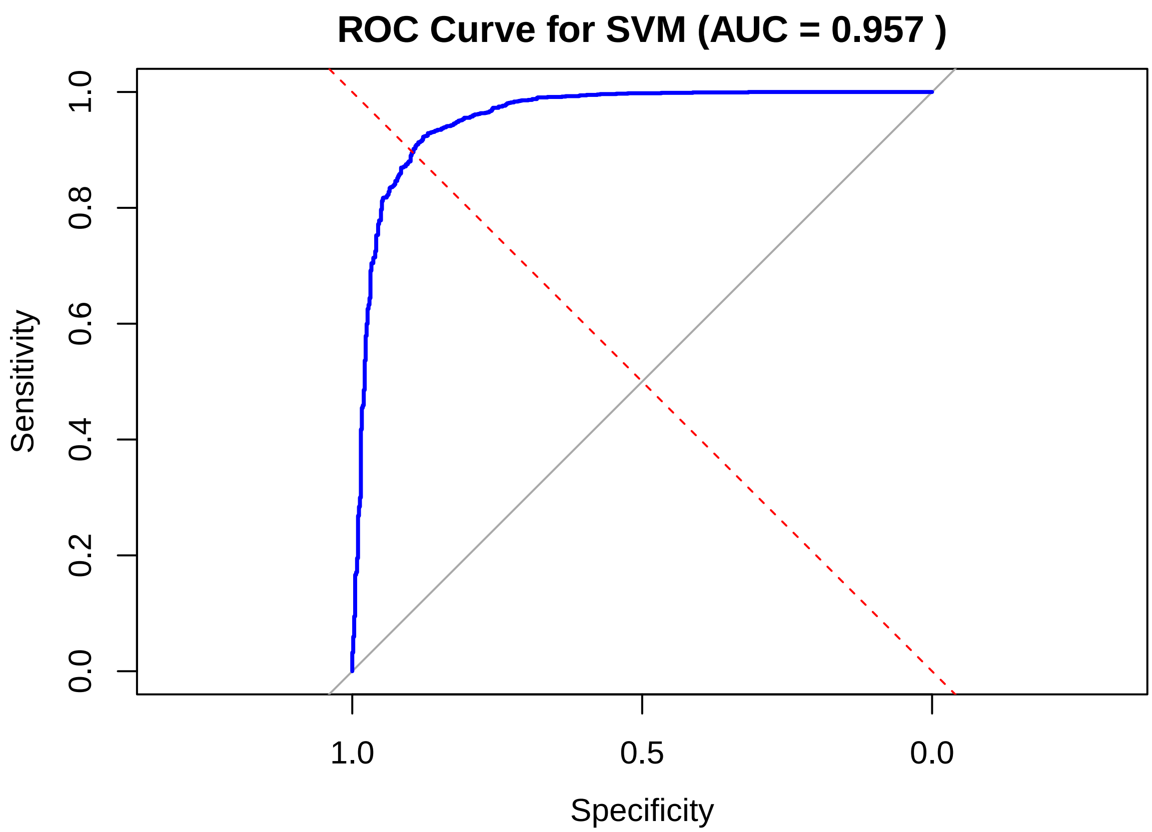


ROC Curve and AUC

Let's evaluate the model's discriminative ability using ROC and AUC:

```r
# Get probability predictions
svm_probs <- predict(svm_final, df.test.svm, probability = TRUE)
svm_probs <- attr(svm_probs, "probabilities")[, "Yes"]
```

```r
# Generate ROC curve
library(pROC)
roc_svm <- roc(df.test.svm$tv, svm_probs)
auc_svm <- auc(roc_svm)

# Plot ROC curve
plot(roc_svm, main = paste("ROC Curve for SVM (AUC =", round(auc_svm, 3), ")"),
     col = "blue", lwd = 2)
abline(a = 0, b = 1, lty = 2, col = "red")
```



Comparison with Other Models

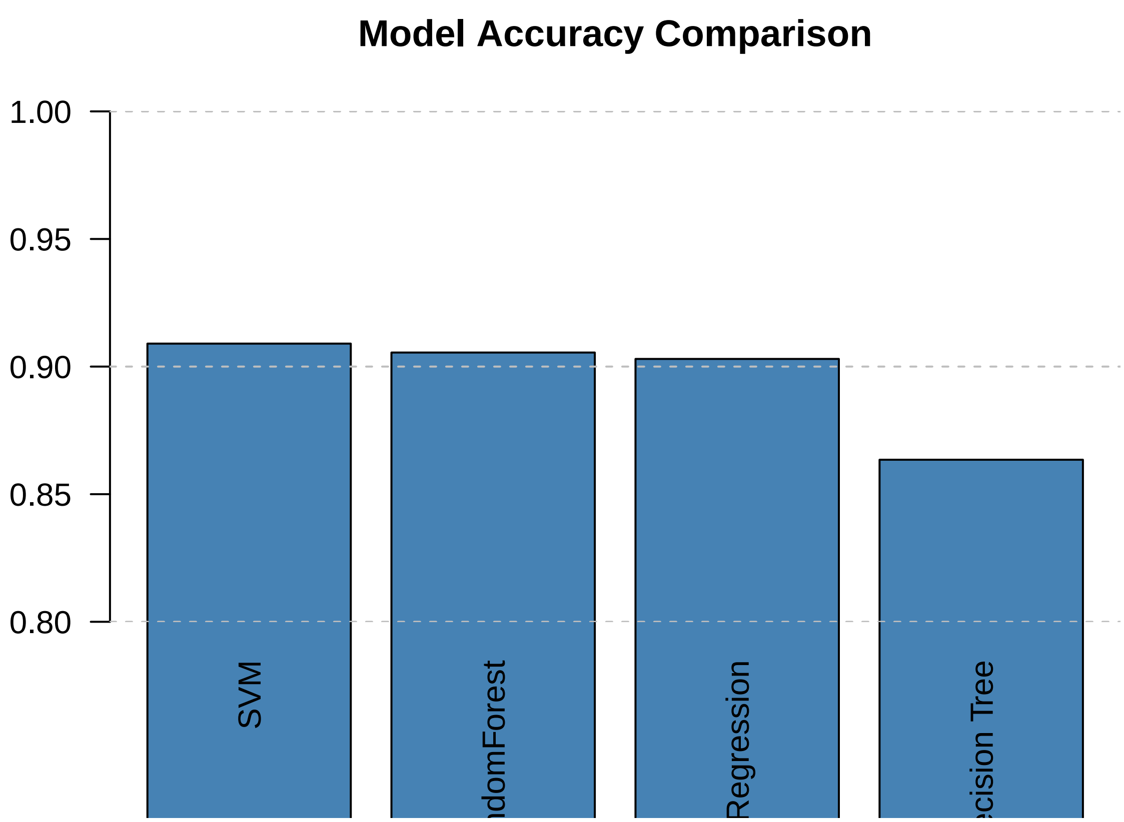
Finally, let's compare the SVM's performance with our previous models:

```r
# Create comparison table for all models (accuracy or error rate)
model_comparison <- data.frame(
  Model = c("Logistic Regression", "Decision Tree", "RandomForest", "SVM"),
  Accuracy = c(
    conf_matrix_test$overall["Accuracy"],  # From Logistic Regression section
    mean(df.tree.pred == df.test$tv),      # From Decision Tree section
    mean(predict(df.rf, df.test) == df.test$tv),  # From Random Forest section
    conf_matrix_final$overall["Accuracy"]  # SVM
  )
)
```

```r
# Sort by accuracy
model_comparison <- model_comparison[order(-model_comparison$Accuracy), ]
print(model_comparison)
```

```
##                 Model Accuracy
## 4                 SVM   0.9090
## 3        RandomForest   0.9055
## 1 Logistic Regression   0.9030
## 2       Decision Tree   0.8635
```

```r
# Visualize model comparison
barplot(model_comparison$Accuracy, names.arg = model_comparison$Model,
        main = "Model Accuracy Comparison", col = "steelblue",
        ylim = c(0.8, 1), las = 2)
abline(h = seq(0, 1, by = 0.1), col = "gray", lty = 2)
```



Interpretation of SVM Results

The Support Vector Machine model for predicting television ownership has shown strong performance:

1. **Model Tuning**: We tuned the cost (C) and gamma parameters through grid search and 5-fold cross-validation. The optimal parameters were found to be cost = 5 and gamma = 0.1, striking a balance between model complexity and generalization ability.

2. **Performance**: The tuned SVM achieved an accuracy of 90.65% on the test set, which is the best out of all the other models (logistic regression, decision tree, and random forest), barely beating out random forest. However given the computationally expensive nature of fitting an SVM model it may be worth considering use of the RF model over the SVM in this situation (only 0.2% accuracy increase from RF).

3. **Avoiding Overfitting/Underfitting**:

   - The learning curve analysis shows that as we increase training data size, the training error increases and test error decreases, indicating the hyperparameters are well tuned because the model doesn't become overfit (training error decreasing and test error increasing) when we include larger portions of the data.
   - The gap between training and test error is roughly 2% (and shrinks with more data), suggesting the model is neither excessively overfit nor underfit.
   - Parameter tuning via cross-validation further ensured optimal regularization.

4. **Discriminative Power**: The AUC of 0.951 demonstrates the model's excellent ability to distinguish between households with and without TVs.

Overall, the SVM model provides a robust prediction of television ownership, balancing complexity with generalization through careful parameter tuning. Its performance compared to the other tested models in this project makes it one of the better models for predicting TV ownership.