```
# ------------------------------------------------
# Project 2: College Admission
# Step 1: Install Packages, Load Data, Check NA, and Transform
# ------------------------------------------------

# Install all necessary packages for the *entire* project
install.packages(c("dplyr", "ggplot2", "caTools", "rpart", "rpart.plot", "e1071"))

# Load libraries
library(dplyr)
library(ggplot2)
library(caTools)
library(rpart)
library(rpart.plot)
library(e1071)

# Load the dataset
file_path <- "/content/College_admission.csv"
df <- read.csv(file_path)
cat("--- Data loaded successfully ---\n\n")


# Predictive Task 1: Find Missing Values (and treat)
# ------------------------------------------------
cat("--- Task 1: Missing Value Check ---\n")
missing_count <- sum(is.na(df))
cat(paste("Total number of missing values (NA):", missing_count, "\n"))

# Treat missing values by removing the rows
df_cleaned <- na.omit(df)
cat(paste("Total rows in original data:", nrow(df), "\n"))
cat(paste("Total rows after NA removal:", nrow(df_cleaned), "\n\n"))


# Predictive Task 3: Find Structure of Data and Transform
# ------------------------------------------------
cat("--- Task 3: Data Structure Transformation ---\n")
cat("Original data structure:\n")
str(df) # Show original structure

# Transform numeric data types to factors (categorical)
# This is crucial for our regression and classification models
df_cleaned$admit <- as.factor(df_cleaned$admit)
df_cleaned$ses <- as.factor(df_cleaned$ses)
df_cleaned$Gender_Male <- as.factor(df_cleaned$Gender_Male)
df_cleaned$Race <- as.factor(df_cleaned$Race)
df_cleaned$rank <- as.factor(df_cleaned$rank)

cat("\n--- Transformed data structure (df_cleaned) ---\n")
str(df_cleaned)
```

```
Installing packages into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependencies ‘bitops’, ‘proxy’



Attaching package: ‘dplyr’


The following objects are masked from ‘package:stats’:

    filter, lag


The following objects are masked from ‘package:base’:
```

```
    intersect, setdiff, setequal, union


Attaching package: ‘e1071’


The following object is masked from ‘package:ggplot2’:

    element


--- Data loaded successfully ---

--- Task 1: Missing Value Check ---
Total number of missing values (NA): 0
Total rows in original data: 400
Total rows after NA removal: 400

--- Task 3: Data Structure Transformation ---
Original data structure:
'data.frame':   400 obs. of  7 variables:
 $ admit      : int  0 1 1 1 0 1 1 0 1 0 ...
 $ gre        : int  380 660 800 640 520 760 560 400 540 700 ...
 $ gpa        : num  3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
 $ ses        : int  1 2 2 1 3 2 2 2 1 1 ...
 $ Gender_Male: int  0 0 0 1 1 1 1 0 1 0 ...
 $ Race       : int  3 2 2 2 2 1 2 2 1 2 ...
 $ rank       : int  3 3 1 4 4 2 1 2 3 2 ...

--- Transformed data structure (df_cleaned) ---
'data.frame':   400 obs. of  7 variables:
 $ admit      : Factor w/ 2 levels "0","1": 1 2 2 2 1 2 2 1 2 1 ...
 $ gre        : int  380 660 800 640 520 760 560 400 540 700 ...
 $ gpa        : num  3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
 $ ses        : Factor w/ 3 levels "1","2","3": 1 2 2 1 3 2 2 2 1 1 ...
 $ Gender_Male: Factor w/ 2 levels "0","1": 1 1 1 2 2 2 2 1 2 1 ...
 $ Race       : Factor w/ 3 levels "1","2","3": 3 2 2 2 2 1 2 2 1 2 ...
 $ rank       : Factor w/ 4 levels "1","2","3","4": 3 3 1 4 4 2 1 2 3 2 ...
```

```
# ---------------------------------------------------
# Project 2: College Admission
# Step 2: Find and Treat Outliers
# ---------------------------------------------------

# We will use the 'df_cleaned' data frame from Step 1

# --- 1. Find Outliers Visually (Boxplots) ---
cat("--- 2.1: Creating Boxplots to Find Outliers ---\n")

# Save boxplot for 'gre'
png("gre_boxplot_before.png")
boxplot(df_cleaned$gre, main="Boxplot for GRE Scores (Before Treatment)", ylab="GRE Score")
dev.off() # Closes the PNG device

# Save boxplot for 'gpa'
png("gpa_boxplot_before.png")
boxplot(df_cleaned$gpa, main="Boxplot for GPA (Before Treatment)", ylab="GPA")
dev.off()

cat("Plots 'gre_boxplot_before.png' and 'gpa_boxplot_before.png' saved.\n")
cat("Check the file explorer to download and view them.\n\n")

# --- 2. Identify Outliers Programmatically ---
cat("--- 2.2: Identifying Outlier Values ---\n")
gre_outliers <- boxplot.stats(df_cleaned$gre)$out
cat("GRE Outliers:\n")
print(gre_outliers)

gpa_outliers <- boxplot.stats(df_cleaned$gpa)$out
cat("\nGPA Outliers:\n")
print(gpa_outliers)
```

```
cat("\n")


# --- 3. Treat Outliers (Capping) ---
# We will cap the outliers by replacing them with the 5th and 95th percentile values.
cat("--- 2.3: Treating Outliers by Capping ---\n")

# Capping for 'gre'
gre_quantiles <- quantile(df_cleaned$gre, probs=c(0.05, 0.95))
gre_low_cap <- gre_quantiles[1]  # 5th percentile
gre_high_cap <- gre_quantiles[2] # 95th percentile

df_cleaned$gre <- ifelse(df_cleaned$gre < gre_low_cap, gre_low_cap, df_cleaned$gre)
df_cleaned$gre <- ifelse(df_cleaned$gre > gre_high_cap, gre_high_cap, df_cleaned$gre)

# Capping for 'gpa'
gpa_quantiles <- quantile(df_cleaned$gpa, probs=c(0.05, 0.95))
gpa_low_cap <- gpa_quantiles[1]  # 5th percentile
gpa_high_cap <- gpa_quantiles[2] # 95th percentile

df_cleaned$gpa <- ifelse(df_cleaned$gpa < gpa_low_cap, gpa_low_cap, df_cleaned$gpa)
df_cleaned$gpa <- ifelse(df_cleaned$gpa > gpa_high_cap, gpa_high_cap, df_cleaned$gpa)

cat("Outliers in 'gre' and 'gpa' have been capped.\n\n")


# --- 4. Verify Treatment (New Boxplots) ---
cat("--- 2.4: Creating Boxplots After Treatment ---\n")

# Save boxplot for 'gre' AFTER
png("gre_boxplot_after.png")
boxplot(df_cleaned$gre, main="Boxplot for GRE Scores (After Treatment)", ylab="GRE Score")
dev.off() # Closes the PNG device

# Save boxplot for 'gpa' AFTER
png("gpa_boxplot_after.png")
boxplot(df_cleaned$gpa, main="Boxplot for GPA (After Treatment)", ylab="GPA")
dev.off()

cat("Plots 'gre_boxplot_after.png' and 'gpa_boxplot_after.png' saved.\n")
cat("View them to see the 'before' and 'after' effect.\n")
```

```
--- 2.1: Creating Boxplots to Find Outliers ---
agg_record_1388154455: 2
agg_record_1388154455: 2
Plots 'gre_boxplot_before.png' and 'gpa_boxplot_before.png' saved.
Check the file explorer to download and view them.

--- 2.2: Identifying Outlier Values ---
GRE Outliers:
[1] 300 300 220 300

GPA Outliers:
[1] 2.26

--- 2.3: Treating Outliers by Capping ---
Outliers in 'gre' and 'gpa' have been capped.

--- 2.4: Creating Boxplots After Treatment ---
agg_record_1388154455: 2
agg_record_1388154455: 2
Plots 'gre_boxplot_after.png' and 'gpa_boxplot_after.png' saved.
View them to see the 'before' and 'after' effect.
```

```
# ------------------------------------------------
# Project 2: College Admission
# Step 3: Check Normal Distribution (Task 4)
# ------------------------------------------------
```

```
# We will use the 'df_cleaned' data frame from Step 2

cat("--- 4.1: Checking Normal Distribution for GRE --- \n")

# 1. Histogram for GRE
png("gre_histogram.png")
hist(df_cleaned$gre, main="Histogram for GRE Scores", xlab="GRE Score", col="skyblue")
dev.off() # Closes the PNG

# 2. Q-Q Plot for GRE
png("gre_qqplot.png")
qqnorm(df_cleaned$gre, main="Q-Q Plot for GRE Scores")
qqline(df_cleaned$gre, col="red", lwd=2) # Adds the reference line
dev.off()

cat("GRE plots 'gre_histogram.png' and 'gre_qqplot.png' saved.\n\n")


cat("--- 4.2: Checking Normal Distribution for GPA --- \n")

# 1. Histogram for GPA
png("gpa_histogram.png")
hist(df_cleaned$gpa, main="Histogram for GPA", xlab="GPA", col="lightgreen")
dev.off()

# 2. Q-Q Plot for GPA
png("gpa_qqplot.png")
qqnorm(df_cleaned$gpa, main="Q-Q Plot for GPA")
qqline(df_cleaned$gpa, col="red", lwd=2)
dev.off()

cat("GPA plots 'gpa_histogram.png' and 'gpa_qqplot.png' saved.\n\n")
```

```
--- 4.1: Checking Normal Distribution for GRE ---
agg_record_1972944858: 2
agg_record_1972944858: 2
GRE plots 'gre_histogram.png' and 'gre_qqplot.png' saved.

--- 4.2: Checking Normal Distribution for GPA ---
agg_record_1972944858: 2
agg_record_1972944858: 2
GPA plots 'gpa_histogram.png' and 'gpa_qqplot.png' saved.
```

```
# ------------------------------------------------
# Project 2: College Admission
# Step 4: Normalize the Data (Task 4 - Part 2)
# ------------------------------------------------

# We will use the 'df_cleaned' data frame from Step 2

cat("--- 4.3: Normalizing 'gre' and 'gpa' --- \n\n")

# Create a min-max normalization function
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

# Apply the normalization function to the 'gre' and 'gpa' columns
# We are overwriting the old columns with the new 0-1 scaled values
df_cleaned$gre <- normalize(df_cleaned$gre)
df_cleaned$gpa <- normalize(df_cleaned$gpa)

cat("--- Data after normalization --- \n")
cat("Note: 'gre' and 'gpa' are now scaled between 0 and 1.\n\n")

# Print a summary of the data
# This will show us the Min (0.00) and Max (1.00) for gre and gpa
print(summary(df_cleaned))
```

```
--- 4.3: Normalizing 'gre' and 'gpa' ---

--- Data after normalization ---
Note: 'gre' and 'gpa' are now scaled between 0 and 1.

 admit        gre              gpa         ses     Gender_Male Race    rank
 0:273   Min.   :0.0000   Min.   :0.0000   1:132   0:210       1:143   1: 61
 1:127   1st Qu.:0.3017   1st Qu.:0.2992   2:139   1:190       2:129   2:151
         Median :0.4514   Median :0.5127   3:129               3:128   3:121
         Mean   :0.4769   Mean   :0.5150                               4: 67
         3rd Qu.:0.6509   3rd Qu.:0.7342
         Max.   :1.0000   Max.   :1.0000
```

```
# ------------------------------------------------
# Project 2: College Admission
# Step 5: Split Data, Run Logistic Regression, Find Significant Variables
# (Tasks 5, 6, and 7)
# ------------------------------------------------

# We will use the 'df_cleaned' data frame from Step 4

# --- 1. Split Data into Training and Testing Sets ---
cat("--- 7.1: Splitting data into Training (70%) and Testing (30%) sets ---\n")

# Set a "seed" so that your random split is the same as mine (reproducible)
set.seed(123)

# Create the split. caTools is already loaded from Step 1
split <- sample.split(df_cleaned$admit, SplitRatio = 0.70)

# Create the two data frames
train_set <- subset(df_cleaned, split == TRUE)
test_set <- subset(df_cleaned, split == FALSE)

cat(paste("Training set size:", nrow(train_set), "\n"))
cat(paste("Testing set size:", nrow(test_set), "\n\n"))


# --- 2. Run Logistic Model (Task 6) & Find Significant Variables (Task 5) ---
cat("--- 5 & 6: Running Full Logistic Model ---\n")

# Build the model using ALL variables
# admit ~ . means "predict admit using all other columns"
model_full <- glm(admit ~ ., data = train_set, family = "binomial")
```

```
model_full <- glm(admit ~ ., data = train_set, family = "binomial")

# Print the summary
cat("--- Summary of Full Model ---\n")
print(summary(model_full))


# --- 3. Calculate Accuracy (Task 7) ---
cat("\n--- 7.2: Calculating Accuracy of Full Model ---\n")

# 1. Make predictions on the TEST data
# type = "response" gives us probabilities (e.g., 0.65)
probabilities <- predict(model_full, newdata = test_set, type = "response")

# 2. Convert probabilities to class predictions (0 or 1)
# If probability > 0.5, predict 1 (Admit), otherwise predict 0 (Don't Admit)
predictions <- ifelse(probabilities > 0.5, 1, 0)

# 3. Create the Confusion Matrix
cat("Confusion Matrix for Full Model:\n")
# We compare the 'predictions' against the actual 'test_set$admit'
conf_matrix <- table(Actual = test_set$admit, Predicted = predictions)
print(conf_matrix)

# 4. Calculate Accuracy
# Accuracy = (Correctly Predicted 0s + Correctly Predicted 1s) / (Total)
#            (Top-Left + Bottom-Right) / (Sum of all)
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
cat(paste("\nModel Accuracy:", accuracy, "\n"))
cat(paste("Model Accuracy (%):", round(accuracy * 100, 2), "%\n"))
```

```
--- 7.1: Splitting data into Training (70%) and Testing (30%) sets ---
Training set size: 280
Testing set size: 120

--- 5 & 6: Running Full Logistic Model ---
--- Summary of Full Model ---

Call:
glm(formula = admit ~ ., family = "binomial", data = train_set)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   -0.1513     0.5443  -0.278 0.781089
gre            1.2162     0.5572   2.183 0.029059 *
gpa            0.9136     0.5146   1.776 0.075813 .
ses2          -0.2980     0.3376  -0.883 0.377388
ses3          -0.7245     0.3514  -2.061 0.039263 *
Gender_Male1  -0.4934     0.2831  -1.743 0.081387 .
Race2         -0.4436     0.3448  -1.287 0.198229
Race3         -0.3472     0.3371  -1.030 0.303122
rank2         -0.5768     0.3881  -1.486 0.137261
rank3         -1.4442     0.4319  -3.344 0.000826 ***
rank4         -1.8504     0.5643  -3.279 0.001040 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 350.14  on 279  degrees of freedom
Residual deviance: 308.47  on 269  degrees of freedom
AIC: 330.47

Number of Fisher Scoring iterations: 4


--- 7.2: Calculating Accuracy of Full Model ---
Confusion Matrix for Full Model:
      Predicted
Actual  0  1
     0 68 14
     1 28 10
```

```
Model Accuracy: 0.65
Model Accuracy (%): 65 %
```

```
# ---------------------------------------------------
# Project 2: College Admission
# Step 6: Run Simpler Logistic Model (Variable Reduction)
# ---------------------------------------------------

cat("--- 6.1: Running Simpler Logistic Model ---\n")
cat("Dropping 'Race' and 'Gender_Male' as they were insignificant.\n\n")

# Build the model using only the significant variables
# We will use the same 'train_set' and 'test_set' from before
model_simple <- glm(admit ~ gre + gpa + ses + rank,
                    data = train_set,
                    family = "binomial")

# Print the summary
cat("--- Summary of Simple Model ---\n")
print(summary(model_simple))


# --- 6.2: Calculate Accuracy of Simple Model ---
cat("\n--- 7.3: Calculating Accuracy of Simple Model ---\n")

# 1. Make predictions on the TEST data
probabilities_simple <- predict(model_simple, newdata = test_set, type = "response")

# 2. Convert probabilities to class predictions
predictions_simple <- ifelse(probabilities_simple > 0.5, 1, 0)

# 3. Create the Confusion Matrix
cat("Confusion Matrix for Simple Model:\n")
conf_matrix_simple <- table(Actual = test_set$admit, Predicted = predictions_simple)
print(conf_matrix_simple)

# 4. Calculate Accuracy
accuracy_simple <- sum(diag(conf_matrix_simple)) / sum(conf_matrix_simple)
cat(paste("\nSimple Model Accuracy:", accuracy_simple, "\n"))
cat(paste("Simple Model Accuracy (%):", round(accuracy_simple * 100, 2), "%\n"))
```

```
--- 6.1: Running Simpler Logistic Model ---
Dropping 'Race' and 'Gender_Male' as they were insignificant.

--- Summary of Simple Model ---

Call:
glm(formula = admit ~ gre + gpa + ses + rank, family = "binomial",
    data = train_set)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.6090     0.4858  -1.254 0.210017
gre           1.1834     0.5491   2.155 0.031145 *
gpa           0.9053     0.5103   1.774 0.076085 .
ses2         -0.4002     0.3289  -1.217 0.223626
ses3         -0.7095     0.3447  -2.058 0.039574 *
rank2        -0.5398     0.3789  -1.425 0.154262
rank3        -1.4210     0.4241  -3.350 0.000807 ***
rank4        -1.7760     0.5528  -3.213 0.001314 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 350.14  on 279  degrees of freedom
Residual deviance: 313.35  on 272  degrees of freedom
AIC: 329.35

Number of Fisher Scoring iterations: 4
```

```
--- 7.3: Calculating Accuracy of Simple Model ---
Confusion Matrix for Simple Model:
      Predicted
Actual  0  1
     0 69 13
     1 26 12

Simple Model Accuracy: 0.675
Simple Model Accuracy (%): 67.5 %
```

```
# ------------------------------------------------
# Project 2: College Admission
# Step 7: Build a Decision Tree Model (Tasks 8 & 9)
# ------------------------------------------------

# The 'rpart' and 'rpart.plot' libraries were already loaded in Step 1

cat("--- 8.1: Building a Decision Tree Model ---\n")

# Build the tree model on the training set
# We use all variables, as trees are good at selecting important ones
tree_model <- rpart(admit ~ ., data = train_set, method = "class")

# Print the text summary of the tree
print(summary(tree_model))
cat("\n")

# --- 8.2: Plotting the Decision Tree ---
cat("--- 8.2: Plotting the Decision Tree ---\n")

# Save the plot to a PNG file
png("decision_tree_plot.png")
rpart.plot(tree_model, main="Decision Tree for College Admission")
dev.off() # Closes the PNG device

cat("Plot 'decision_tree_plot.png' has been saved to your Colab files.\n")
cat("This plot shows the 'rules' the model learned.\n\n")


# --- 9.1: Calculate Decision Tree Accuracy ---
cat("--- 9.1: Calculating Accuracy of Decision Tree Model ---\n")

# 1. Make predictions on the TEST data
# type = "class" gives the final prediction (0 or 1)
tree_predictions <- predict(tree_model, newdata = test_set, type = "class")

# 2. Create the Confusion Matrix
cat("Confusion Matrix for Decision Tree Model:\n")
tree_conf_matrix <- table(Actual = test_set$admit, Predicted = tree_predictions)
print(tree_conf_matrix)

# 3. Calculate Accuracy
tree_accuracy <- sum(diag(tree_conf_matrix)) / sum(tree_conf_matrix)
cat(paste("\nDecision Tree Model Accuracy:", tree_accuracy, "\n"))
cat(paste("Decision Tree Model Accuracy (%):", round(tree_accuracy * 100, 2), "%\n"))
```

```
--- 8.1: Building a Decision Tree Model ---
Call:
rpart(formula = admit ~ ., data = train_set, method = "class")
  n= 280

          CP nsplit rel error   xerror       xstd
1 0.05992509      0 1.0000000 1.000000 0.08754728
2 0.03370787      3 0.8202247 1.011236 0.08780698
3 0.02247191      4 0.7865169 1.000000 0.08754728
4 0.01123596      5 0.7640449 1.044944 0.08855096
5 0.01000000      7 0.7415730 1.123596 0.09008806

Variable importance
       rank         gpa         gre Gender_Male         ses        Race
         31          26          16          16           7           5

Node number 1: 280 observations,    complexity param=0.05992509
  predicted class=0  expected loss=0.3178571  P(node) =1
    class counts:   191    89
   probabilities: 0.682 0.318
  left son=2 (127 obs) right son=3 (153 obs)
  Primary splits:
      rank        splits as  RRLL, improve=7.721053, (0 missing)
      gre         < 0.6259352  to the left,  improve=5.065907, (0 missing)
      gpa         < 0.6093435  to the left,  improve=4.118571, (0 missing)
      ses         splits as  RLL, improve=1.338388, (0 missing)
      Gender_Male splits as  RL, improve=1.017252, (0 missing)
  Surrogate splits:
      gre < 0.07730673 to the left,  agree=0.579, adj=0.071, (0 split)
      gpa < 0.1421667  to the left,  agree=0.557, adj=0.024, (0 split)

Node number 2: 127 observations
  predicted class=0  expected loss=0.1889764  P(node) =0.4535714
    class counts:   103    24
   probabilities: 0.811 0.189

Node number 3: 153 observations,    complexity param=0.05992509
  predicted class=0  expected loss=0.4248366  P(node) =0.5464286
    class counts:    88    65
   probabilities: 0.575 0.425
  left son=6 (77 obs) right son=7 (76 obs)
  Primary splits:
      gpa         < 0.5247684  to the left,  improve=3.9691230, (0 missing)
      gre         < 0.7755611  to the left,  improve=3.4840240, (0 missing)
      Gender_Male splits as  RL, improve=1.9965940, (0 missing)
      rank        splits as  RL--, improve=1.1341760, (0 missing)
      Race        splits as  RLL, improve=0.5355516, (0 missing)
  Surrogate splits:
      gre         < 0.5760599  to the left,  agree=0.595, adj=0.184, (0 split)
      ses         splits as  LLR, agree=0.523, adj=0.039, (0 split)
      rank        splits as  RL--, agree=0.516, adj=0.026, (0 split)
      Gender_Male splits as  LR, agree=0.510, adj=0.013, (0 split)

Node number 6: 77 observations,    complexity param=0.03370787
  predicted class=0  expected loss=0.3116883  P(node) =0.275
    class counts:    53    24
   probabilities: 0.688 0.312
  left son=12 (70 obs) right son=13 (7 obs)
  Primary splits:
      gre  < 0.8004988  to the left,  improve=2.496104000, (0 missing)
      ses  splits as  RLL, improve=1.762943000, (0 missing)
      Race splits as  RLL, improve=1.465628000, (0 missing)
      gpa  < 0.1663311  to the left,  improve=0.674048800, (0 missing)
      rank splits as  LR--, improve=0.007382092, (0 missing)

Node number 7: 76 observations,    complexity param=0.05992509
  predicted class=1  expected loss=0.4605263  P(node) =0.2714286
    class counts:    35    41
   probabilities: 0.461 0.539
  left son=14 (36 obs) right son=15 (40 obs)
  Primary splits:
      Gender_Male splits as  RL, improve=4.352047, (0 missing)
      rank        splits as  RL--, improve=2.184033, (0 missing)
      gre         < 0.2518703  to the left,  improve=2.055032, (0 missing)
      gpa         < 0.593234   to the right, improve=1.263158, (0 missing)
```

```
      ses         splits as  LLR, improve=0.406015, (0 missing)
  Surrogate splits:
      gpa  < 0.9154249  to the right, agree=0.592, adj=0.139, (0 split)
      gre  < 0.5760599  to the right, agree=0.579, adj=0.111, (0 split)
      rank splits as  RL--, agree=0.553, adj=0.056, (0 split)

Node number 12: 70 observations,    complexity param=0.01123596
  predicted class=0  expected loss=0.2714286  P(node) =0.25
    class counts:    51    19
   probabilities: 0.729 0.271
  left son=24 (45 obs) right son=25 (25 obs)
  Primary splits:
      Race        splits as  RLL, improve=1.28571400, (0 missing)
      ses         splits as  RLL, improve=1.21601700, (0 missing)
      gre         < 0.5760599  to the right, improve=0.44144980, (0 missing)
      gpa         < 0.1703584  to the left,  improve=0.40491520, (0 missing)
      Gender_Male splits as  RL, improve=0.01987475, (0 missing)
  Surrogate splits:
      gpa < 0.4482481  to the left,  agree=0.714, adj=0.2, (0 split)

Node number 13: 7 observations
  predicted class=1  expected loss=0.2857143  P(node) =0.025
    class counts:     2     5
   probabilities: 0.286 0.714

Node number 14: 36 observations,    complexity param=0.02247191
  predicted class=0  expected loss=0.3611111  P(node) =0.1285714
    class counts:    23    13
   probabilities: 0.639 0.361
  left son=28 (28 obs) right son=29 (8 obs)
  Primary splits:
      gpa  < 0.947644   to the left,  improve=1.4325400, (0 missing)
      rank splits as  RL--, improve=1.4325400, (0 missing)
      ses  splits as  LLR, improve=0.4104422, (0 missing)
      Race splits as  RLL, improve=0.4104422, (0 missing)
      gre  < 0.6259352  to the left,  improve=0.3361111, (0 missing)

Node number 15: 40 observations
  predicted class=1  expected loss=0.3  P(node) =0.1428571
    class counts:    12    28
   probabilities: 0.300 0.700

Node number 24: 45 observations
  predicted class=0  expected loss=0.2  P(node) =0.1607143
    class counts:    36     9
   probabilities: 0.800 0.200

Node number 25: 25 observations,    complexity param=0.01123596
  predicted class=0  expected loss=0.4  P(node) =0.08928571
    class counts:    15    10
   probabilities: 0.600 0.400
  left son=50 (11 obs) right son=51 (14 obs)
  Primary splits:
      ses         splits as  RRL, improve=1.87013000, (0 missing)
      gpa         < 0.4603302  to the right, improve=0.52941180, (0 missing)
      gre         < 0.4264339  to the right, improve=0.11688310, (0 missing)
      Gender_Male splits as  LR, improve=0.01282051, (0 missing)
  Surrogate splits:
      gpa  < 0.4603302  to the right, agree=0.72, adj=0.364, (0 split)
      rank splits as  RL--, agree=0.68, adj=0.273, (0 split)
      gre  < 0.1022444  to the left,  agree=0.60, adj=0.091, (0 split)

Node number 28: 28 observations
  predicted class=0  expected loss=0.2857143  P(node) =0.1
    class counts:    20     8
   probabilities: 0.714 0.286

Node number 29: 8 observations
  predicted class=1  expected loss=0.375  P(node) =0.02857143
    class counts:     3     5
   probabilities: 0.375 0.625

Node number 50: 11 observations
  predicted class=0  expected loss=0.1818182  P(node) =0.03928571
    class counts:     9     2
   probabilities: 0.818 0.182
```