Data Analytics & Predictive Modeling

Lab#4, Fall 2023

The dataset for this lab ("GA_Pred.csv") contains several parameters that are important for graduate admission in the US. Your objective is to use this dataset to analyze and predict the relationship between these parameters and the chance of admission.

1. (10 points) Name this dataset as "GA_Pred_C". Pick a method to identify outliers for each numerical variable, and explain this method. Document your findings in the report.

*For the following questions use "GA_Pred_C" unless otherwise mentioned.

In order to find outliers, I am first validating if the data has Nan values.

R code:

```r
# total number of missing values
count_na <- sum(is.na(GA_Pred_C))
print(paste("Total Number of missing values in the dataframe: ", count_na))

# total number of missing values in each column
sapply(GA_Pred_C, function(col) sum(is.na(col)))
```

Output

```
> print(paste("Total Number of missing values in the dataframe: ", count_na))
[1] "Total Number of missing values in the dataframe:  4"
> 
```

```
> sapply(GA_Pred_C, function(col) sum(is.na(col)))
       Serial.No.         GRE.Score       TOEFL.Score University.Rating               SOP               LOR              CGPA 
                0                 0                 0                 1                 1                 1                 0 
         Research   Chance.of.Admit 
                1                 0 
```

We can see from the above output that 4 columns contains nan values. Out of 4 columns, 2 are numerical columns and 2 are categorical columns. I am imputing nan in numerical columns with mean and nan in categorical columns with mode.

R code:

```r
numeric_columns <- c("GRE.Score", "TOEFL.Score", "SOP", "LOR", "CGPA", "Chance.of.Admit")
categorical_columns <- c("University.Rating", "Research")

na_columns <- colSums(is.na(GA_Pred_C[, numeric_columns]))
columns_with_na <- names(na_columns[na_columns > 0])
cat("Numeric Columns with NA values:", toString(columns_with_na), "\n")
for(col in columns_with_na){
  # replace the numeric column with mean
  GA_Pred_C[[col]][is.na(GA_Pred_C[[col]])] <- mean(GA_Pred_C[[col]], na.rm=TRUE)
}

na_columns <- colSums(is.na(GA_Pred_C[, categorical_columns]))
columns_with_na <- names(na_columns[na_columns > 0])
cat("Categorical Columns with NA values:", toString(columns_with_na), "\n")
for(col in columns_with_na){
  # replace the numeric column with mode
  GA_Pred_C[[col]][is.na(GA_Pred_C[[col]])] <- mode(GA_Pred_C[[col]])
}

sapply(GA_Pred_C, function(col) sum(is.na(col)))
```

Output:

```
> sapply(GA_Pred_C, function(col) sum(is.na(col)))
       Serial.No.         GRE.Score       TOEFL.Score University.Rating               SOP               LOR              CGPA
                0                 0                 0                 0                 0                 0                 0
         Research     Chance.of.Admit
                0                 0
>
```

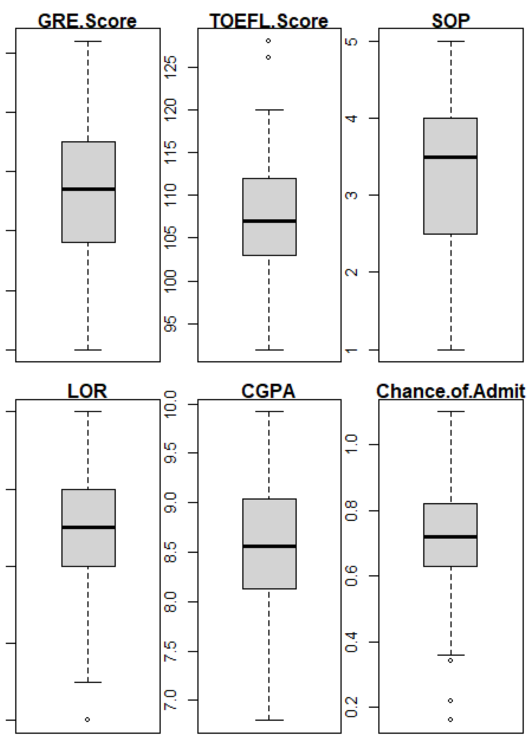
About output presents that there are no nan values in the data.

Box Plots: Box plots give an illustration of the data distribution. Points outside of the box plot's "whiskers" are known as outliers.

R code:

```r
#Q1

par(mfrow = c(2, 3), mar=c(1,1,1,1))
for (col in numeric_columns) {
  boxplot(GA_Pred_C[[col]], main = col, ylab = col)
}
```

Output:



Z-Score: The Z-score calculates a data point's deviation from the mean in standard deviations. An outlier is usually indicated by a z-score larger than a 3 standard deviation as threshold.

$Z=(X-\mu)/\sigma$

R Code:

```r
# 3 standard deviation as threshold
outlier_threshold <- 3

# store outliers for each variable
outliers_list <- list()

for (col in numeric_columns) {
  # z-scores
  z_scores <- scale(GA_Pred_C[[col]])

  # upper and lower thresholds
  upper_threshold <- mean(z_scores) + outlier_threshold * sd(z_scores)
  lower_threshold <- mean(z_scores) - outlier_threshold * sd(z_scores)

  # outliers
  outliers <- which(abs(z_scores) > outlier_threshold)

  outliers_list[[col]] <- GA_Pred_C[outliers, ]
}

for (col in numeric_columns) {
  cat("Outliers for", col, ":\n")
  print(outliers_list[[col]])
  cat("\n")
}
```

Output:

```
Outliers for GRE.Score :
[1] Serial.No.        GRE.Score         TOEFL.Score       University.Rating SOP               LOR               CGPA
[8] Research          Chance.of.Admit
<0 rows> (or 0-length row.names)

Outliers for TOEFL.Score :
   Serial.No. GRE.Score TOEFL.Score University.Rating SOP LOR CGPA Research Chance.of.Admit
25         25       336         128                 5 4.0 3.5 9.80        1            0.97
96         96       304         126                 4 1.5 2.5 7.84        0            0.42

Outliers for SOP :
[1] Serial.No.        GRE.Score         TOEFL.Score       University.Rating SOP               LOR               CGPA
[8] Research          Chance.of.Admit
<0 rows> (or 0-length row.names)

Outliers for LOR :
[1] Serial.No.        GRE.Score         TOEFL.Score       University.Rating SOP               LOR               CGPA
[8] Research          Chance.of.Admit
<0 rows> (or 0-length row.names)

Outliers for CGPA :
[1] Serial.No.        GRE.Score         TOEFL.Score       University.Rating SOP               LOR               CGPA
[8] Research          Chance.of.Admit
<0 rows> (or 0-length row.names)

Outliers for Chance.of.Admit :
    Serial.No. GRE.Score TOEFL.Score University.Rating SOP LOR CGPA Research Chance.of.Admit
235        235       330         113                 5   5 4.0 9.31        1            0.16
274        274       312          99                 1   1 1.5 8.01        1            0.22
```

From the above output, we can see that there are outliers for TOEFL score and chance of admit.

2. (10 points) Apply polynomial regression with d(degree) chosen by cross-validation in order to predict the chance of admission using GRE score. Plot the resulting polynomial fit to the data. Comment on the selected d and the plot.

In polynomial regression, an n-th degree polynomial is used to model the relationship between the independent variable (x) and the dependent variable (y).

It applies a polynomial equation to the data in order to capture more intricate relationships.

One of the most important aspects of polynomial regression is choosing the degree of the polynomial (n), which is typically done using methods like cross-validation to identify the model that strikes the best balance between complexity and data fit.

Step 1: I am splitting the data into 80-20 ratio. 80% of data is train set and 20% of the data is test set.

```
#Q2
# Sample split for training and testing
train_ratio <- 0.8
split <- sample.split(GA_Pred_C$Chance.of.Admit, SplitRatio = train_ratio)

# Create training and testing datasets
train_data <- subset(GA_Pred_C, split == TRUE)
test_data <- subset(GA_Pred_C, split == FALSE)

print(paste("Train data size: ", nrow(train_data), ncol(train_data)))
print(paste("Test data size: ", nrow(test_data), ncol(test_data)))
```

Output:

```
> print(paste("Train data size: ", nrow(train_data), ncol(train_data)))
[1] "Train data size:  400 9"
> print(paste("Test data size: ", nrow(test_data), ncol(test_data)))
[1] "Test data size:  100 9"
```

Step 2: I am performing cross validation to find the optimal degree for polynomial regression.

In cross-validation, the dataset is divided into several subsets, the model is trained on some of these subsets, and its performance is assessed on the remaining data. To determine the ideal degree, this process is repeated, averaging performance over various subsets.

The optimal degree of freedom is 2.

R code:

```r
# Cross-validation to find the optimal degree for polynomial regression
max_degree <- 10
cv_error <- rep(0, max_degree)

for (degree in 1:max_degree) {
  model_formula <- as.formula(paste("Chance.of.Admit ~ poly(GRE.Score, ", degree, ")", sep = ""))
  cv_error[degree] <- cv.glm(train_data, glm(model_formula, data = train_data), K = 10)$delta[1]
}

optimal_degree <- which.min(cv_error)
cat("The optimal degree for the polynomial regression model is:", optimal_degree, "\n")
```

Output:

```
> cat("The optimal degree for the polynomial regression model is:", optimal_degree, "\n")
The optimal degree for the polynomial regression model is: 2
```

Step 3: I am training the polynomial regression model with the optimal degree of freedom and then making the predictions on the test data. Evaluating the model's performance with RMSE.

R code

```r
# Fit the tuned polynomial regression model with the optimal degree
tuned_model <- lm(Chance.of.Admit ~ poly(GRE.Score, optimal_degree), data = train_data)

# Make predictions on both training and test sets
train_pred <- predict(tuned_model, newdata = list(GRE.Score = train_data$GRE.Score))
test_pred <- predict(tuned_model, newdata = list(GRE.Score = test_data$GRE.Score))

# Compute the root mean squared error (RMSE) of the predictions
train_rmse <- sqrt(mean((train_pred - train_data$Chance.of.Admit)^2))
test_rmse <- sqrt(mean((test_pred - test_data$Chance.of.Admit)^2))

cat("Train RMSE with the optimal degree for the polynomial regression model is:", train_rmse, "\n")
cat("Test RMSE with the optimal degree for the polynomial regression model is:", test_rmse, "\n")
```

Output:

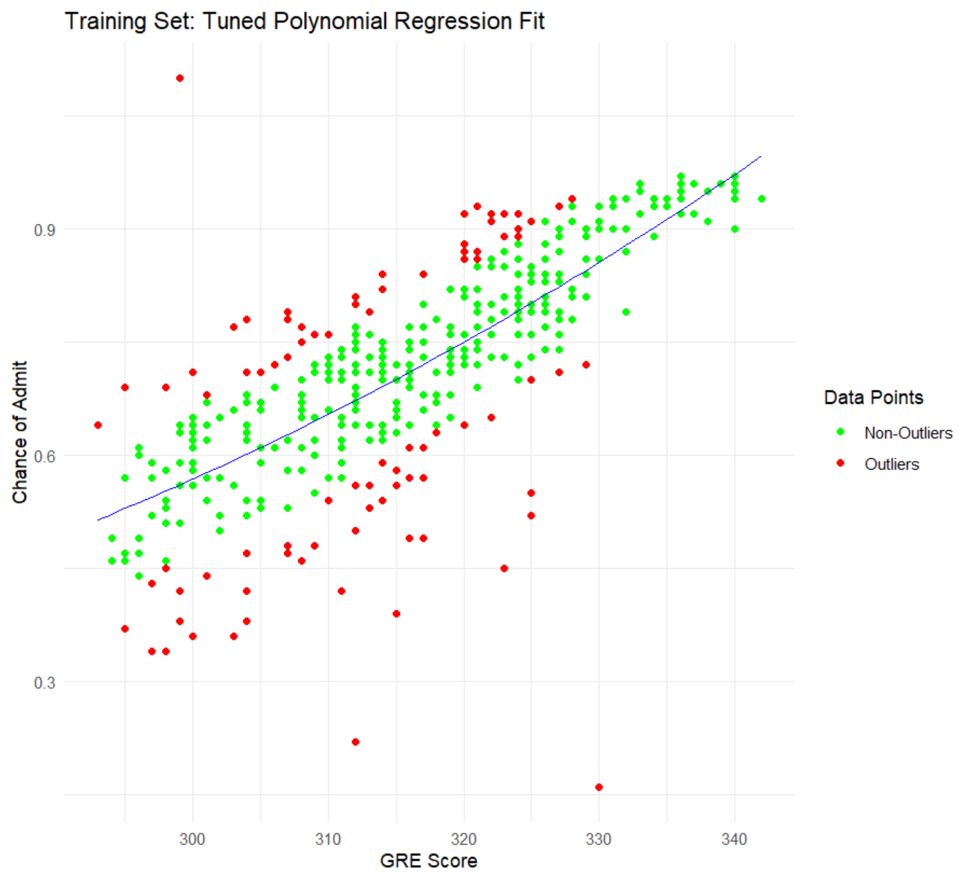
```
> cat("Train RMSE with the optimal degree for the polynomial regression model is:", train_rmse, "\n")
Train RMSE with the optimal degree for the polynomial regression model is: 0.09840323
> cat("Test RMSE with the optimal degree for the polynomial regression model is:", test_rmse, "\n")
Test RMSE with the optimal degree for the polynomial regression model is: 0.07121853
```

Step 4: Plotting the model fit on train data and the predictions.

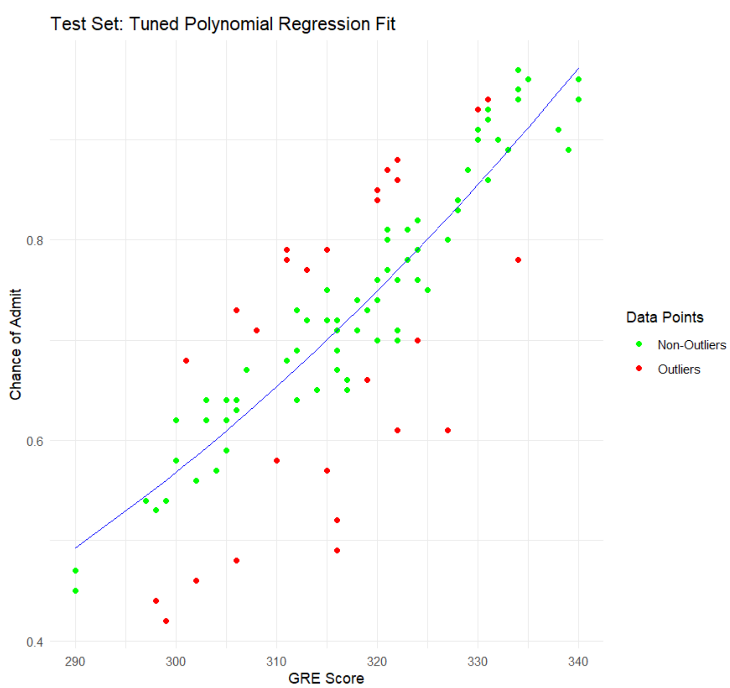
The polynomial degree of two selected for the plot is suitable for the given data. The graph clearly demonstrates a parabolic relationship between the likelihood of admission and the GRE score. This relationship can be effectively represented by a quadratic polynomial regression model.

The following are some benefits of utilising a regression model with quadratic polynomials:

- **Non-linear relationships between the variables can be captured by it.**
- **It is not too difficult to understand and use.**
- **Computationally, it is effective.**



Training Set: Tuned Polynomial Regression Fit
0.9
0.6
0.3
Chance of Admit
300
310
320
330
340
GRE Score
Data Points
Non-Outliers
Outliers



Test Set: Tuned Polynomial Regression Fit
0.8
0.6
0.4
Chance of Admit
290
300
310
320
330
340
GRE Score
Data Points
Non-Outliers
Outliers

3. (20 points) Smoothing splines:

I. Use a smoothing spline with df and lambda chosen by cross-validation in order to fit the chance of admission to the TOEFL scores. Plot the fitted spline on the data, and comment on the plot. (You may use splines package and cv=TRUE for cross-validation)

The non-linear relationship between TOEFL scores and the chance of admission is indicated by the fitted smoothing spline displayed in the image. The initial concavity of the curve suggests that the likelihood of admission rises quickly with a higher TOEFL score. At higher TOEFL scores, the curve starts to flatten out, indicating that the likelihood of admission reaches a plateau.

The fitted smoothing spline offers a decent fit for the data. In addition to accounting for data outliers, it captures the non-linear relationship between TOEFL scores and admission chance.

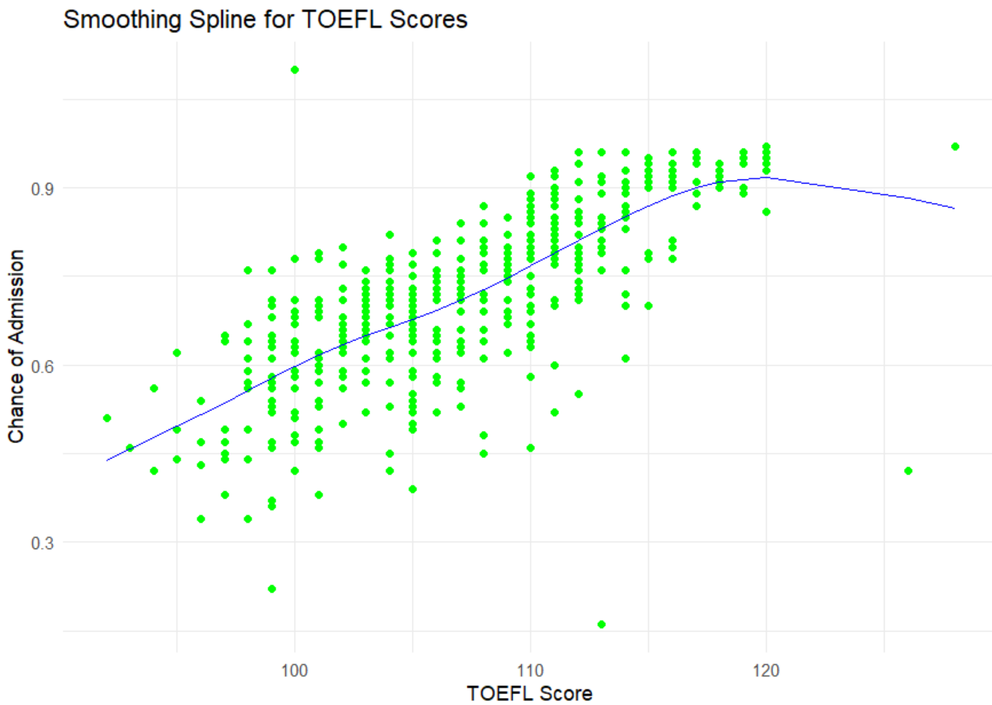
The above graph's smoothing parameter, 0.0126916, is a reasonable choice. Without overfitting, the curve is smooth and provides a good fit to the data.

The trade-off between smoothing out the noise and fitting the data is managed by the smoothing parameter. A smoother curve will be produced by increasing the smoothing parameter, but the data may lose some significant details as a result. A wigglier curve will be produced by a smaller smoothing parameter, but this may be overfitting the data's noise.

R code:

```
#Q3

# applying smooth spline
smoothend_spline <- smooth.spline(GA_Pred_C$TOEFL.Score,
                                  GA_Pred_C$Chance.of.Admit, cv = TRUE)

# Create a data frame for plotting
plot_data <- data.frame(TOEFL.Score = GA_Pred_C$TOEFL.Score,
                        Chance.of.Admit = GA_Pred_C$Chance.of.Admit)

# Plot the data and the fitted spline using ggplot2
ggplot(plot_data, aes(x = TOEFL.Score, y = Chance.of.Admit)) +
  geom_point(col = "green") +
  geom_line(aes(y = fitted(smoothend_spline)), color = "blue") +
  labs(x = "TOEFL Score", y = "Chance of Admission") +
  ggtitle("Smoothing Spline for TOEFL Scores") +
  theme_minimal() +
  theme(legend.position = "topright") +
  guides(col = guide_legend(title = "Smoothing Spline"))

# Print the chosen values of df and lambda
cat("Degrees of Freedom (df):", smoothend_spline$df, "\n")
cat("Smoothing Parameter (lambda):", smoothend_spline$lambda, "\n")

```

Output:




```
> cat("Degrees of Freedom (df):", smoothend_spline$df, "\n")
Degrees of Freedom (df): 5.32607
> cat("Smoothing Parameter (lambda):", smoothend_spline$lambda, "\n")
Smoothing Parameter (lambda): 0.0126916
```

II. Discuss how the values of lambda and df affect smoothing splines.

The main function of λ is to manage the trade-off between obtaining a smooth curve and closely fitting the data. A smoother curve is associated with a larger because it penalizes deviations from a simple function or straight line.

Lower values of λ enable the spline to closely track the data points, which may cause noise to be captured and overfitting to occur. Greater generalization is facilitated by larger values of λ, which lessen the influence of individual data points and prevent overfitting.

A smoothing spline's degrees of freedom show how flexible the model is. The spline can be more flexible and capture minute details in the data with a higher df. On the other hand, a smoother, less flexible curve is produced by a lower df.

Increasing df usually makes the model more accurate at fitting the data, but it can also cause overfitting. Achieving a balance between preserving simplicity and identifying the underlying trend is crucial.

Selecting the right value for λ and df requires striking a compromise between preventing needless complexity and providing a good fit for the data. The best values can be found with the aid of model selection methods such as cross-validation.

A small and a high df could cause overfitting and noise capture, while a large λ and a low df could lead to an oversimplified model that misses significant patterns.

By resolving a penalized least squares issue, smoothing splines are fitted. Higher df values result in an increase in computational cost, while larger λ values can simplify the optimization problem and increase its computational efficiency.

4. (20 points) Assume that we are interested in predicting whether a student's chance of acceptance is above 0.72. Split the dataset into 80% train and 20% test. Use the train set and fit a polynomial logistic regression model using GRE score with the optimal d obtained in question 2 and CGPA with d=1. Apply the fitted model to the test dataset to predict the chance of admission. Plot the predicted probabilities versus actual probabilities and discuss the performance of the fitted model. (Use type="response" in predict()).

Step 1: I am using the optimal degree for GRE as 2 (obtained in q2) and for CGPA is 1. I am fitting the logistic regression model on train data and testing on test data. Chance of acceptance above 0.72 is 1 and below or equal to 0.72 is 0.

R code:

```
#Q4
optimal_degree_GRE <- 2 #optimal degree from Q2
optimal_degree_CGPA <- 1

# Create formula for logistic regression
formula <- as.formula(paste("Chance.of.Admit > 0.72 ~ poly(GRE.Score, ",
                            optimal_degree_GRE, ") + poly(CGPA, ", optimal_degree_CGPA, ")"))

# Fit logistic regression model
logistic_model <- glm(formula, data = train_data, family = binomial)

# Predict on the test set
predicted_probabilities <- predict(logistic_model, newdata = test_data, type = "response")

test_rmse <- sqrt(mean((predicted_probabilities - test_data$Chance.of.Admit)^2))
cat(test_rmse)

```

Output

```
> cat(test_rmse)
0.3456252
```

Step 2: To evaluate the model, I am first using 0.50 as the threshold on predicted probabilities which is the standard way of doing.

R code:

```r
# Evaluate model performance
threshold <- 0.5
predicted_labels <- ifelse(predicted_probabilities > threshold, 1, 0)
actual_labels <- ifelse(test_data$Chance.of.Admit > 0.72, 1, 0)

# Confusion matrix
conf_matrix <- table(Actual = actual_labels, Predicted = predicted_labels)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
cat("Accuracy:", accuracy, "\n")
```

Output:

```
> print("Confusion Matrix:")
[1] "Confusion Matrix:"
> print(conf_matrix)
      Predicted
Actual  0  1
     0 39  9
     1  9 43

> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.82
```

Step 3: I am using 0.72 as the threshold on predicted probabilities.

R code:

```r
# Evaluate model performance
threshold <- 0.72
predicted_labels <- ifelse(predicted_probabilities > threshold, 1, 0)
actual_labels <- ifelse(test_data$Chance.of.Admit > 0.72, 1, 0)

# Confusion matrix
conf_matrix <- table(Actual = actual_labels, Predicted = predicted_labels)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
cat("Accuracy:", accuracy, "\n")
```

Output:

```
[1] "Confusion Matrix:"
> print(conf_matrix)
      Predicted
Actual  0  1
     0 45  3
     1 12 40
```

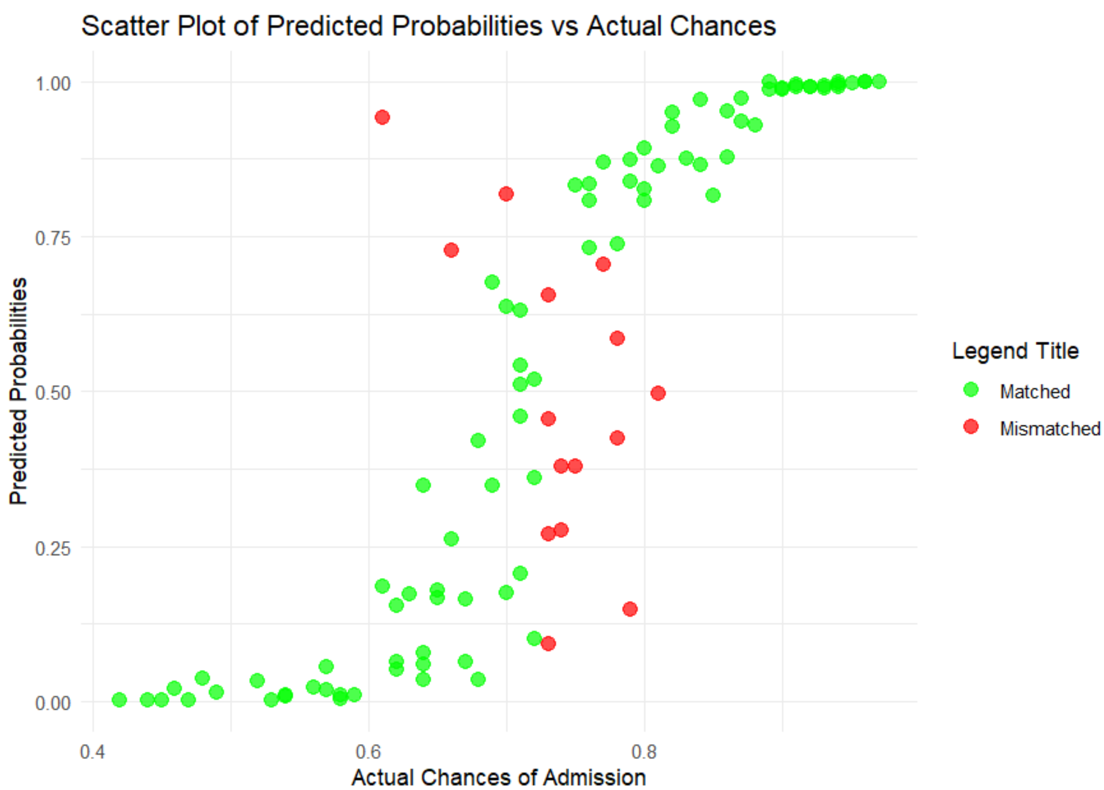
```
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.85
```

Step 4: Plotting the fitted model.

R code:

```r
# Create a data frame for plotting
plot_data <- data.frame(
  Actual_Chance = test_data$Chance.of.Admit,
  Predicted_Probabilities = predicted_probabilities,
  Color = ifelse((predicted_probabilities > 0.72 & test_data$Chance.of.Admit <= 0.72) |
                   (predicted_probabilities <= 0.72 & test_data$Chance.of.Admit > 0.72), "red", "green")
)

# Plot the scatter plot with color-coded points for mismatched probabilities
library(ggplot2)

ggplot(plot_data, aes(x = Actual_Chance, y = Predicted_Probabilities, color = Color)) +
  geom_point(alpha = 0.7, size = 3) +  # Fix: alpha should be between 0 and 1
  ggtitle("Scatter Plot of Predicted Probabilities vs Actual Chances") +
  xlab("Actual Chances of Admission") +
  ylab("Predicted Probabilities") +
  scale_color_manual(values = c("green" = "green", "red" = "red"), name = "Legend Title",
                     labels = c("Matched", "Mismatched")) +
  theme_minimal()
```

Output:



The graph displays a logistic regression model's scatter plot of expected probabilities versus actual admission chances. The diagonal line occupies the majority of the scatter plot, suggesting that the model is well-calibrated. This indicates that the expected and actual chances of admission are fairly close to each other.

A small number of the data points are situated below the diagonal. This indicates that these students' actual chances of admission were higher than the model's predicted probability of admission. Several things could be the cause of this, including:

- **It's possible that the model did not account for every significant factor influencing admissions decisions.**
- **It's possible that the model is overfitting the training set, which would explain why it performs poorly when applied to fresh data.**
- **Predictions may be off if there is noise in the data.**

5. (20 points) Split the dataset into 70% train and 30% test. Fit a MARS on the training and use that model to predict the test data set. Plot the predicted probabilities versus actual probabilities and discuss the performance of the fitted model. (Use “mda” or “earth” package)

Step 1: I am splitting the data with 70-30 % ratio.

R code:

```
#Q5

# Sample split for training and testing
train_ratio <- 0.8
split <- sample.split(GA_Pred_C$Chance.of.Admit, SplitRatio = train_ratio)

# Create training and testing datasets
train_data_Q5 <- subset(GA_Pred_C, split == TRUE)
test_data_Q5 <- subset(GA_Pred_C, split == FALSE)

print(paste("Train data size: ", nrow(train_data_Q5), ncol(train_data_Q5)))
print(paste("Test data size: ", nrow(test_data_Q5), ncol(test_data_Q5)))
```

Output:

```
> print(paste("Train data size: ", nrow(train_data_Q5), ncol(train_data_Q5)))
[1] "Train data size:  400 9"
> print(paste("Test data size: ", nrow(test_data_Q5), ncol(test_data_Q5)))
[1] "Test data size:  100 9"
```

Step 2: Fitting the MARS on the train data and predicting on test data. Evaluating the model’s performance by considering 0.50 as threshold.

R code:

```r
mars_model <- earth(Chance.of.Admit ~ GRE.Score +TOEFL.Score+ CGPA, data = train_data_Q5)

# Predict on the test set
predicted_probabilities <- predict(mars_model, newdata = test_data_Q5)

# Evaluate model performance
threshold <- 0.5
predicted_labels <- ifelse(predicted_probabilities > threshold, 1, 0)
actual_labels <- ifelse(test_data_Q5$Chance.of.Admit > 0.72, 1, 0)

# Confusion matrix
conf_matrix <- table(Actual = actual_labels, Predicted = predicted_labels)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
cat("Accuracy:", accuracy, "\n")
```

Output:

```
> print(conf_matrix)
      Predicted
Actual  0  1
     0  5 43
     1  0 52

> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.57
```

Step 3: Evaluating the model's performance by considering 0.72 as threshold.

R code:

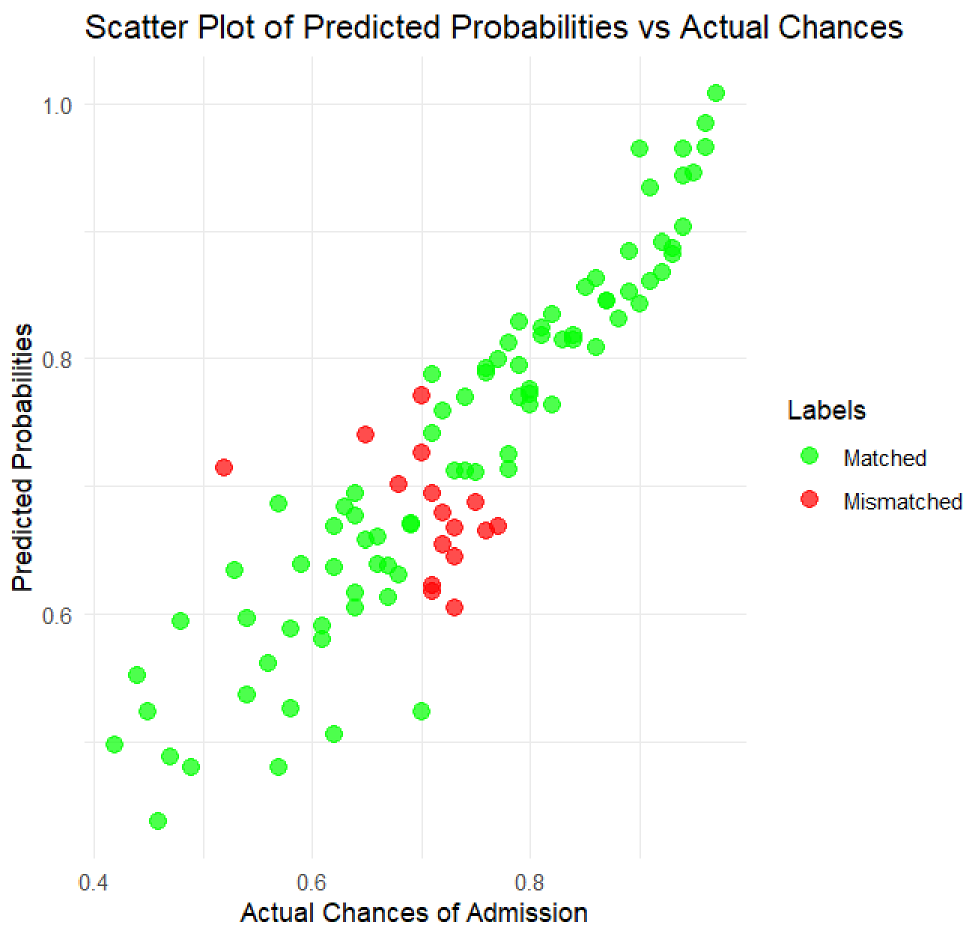
```r
threshold <- 0.72
predicted_labels <- ifelse(predicted_probabilities > threshold, 1, 0)
actual_labels <- ifelse(test_data_Q5$Chance.of.Admit > 0.72, 1, 0)

# Confusion matrix
conf_matrix <- table(Actual = actual_labels, Predicted = predicted_labels)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
cat("Accuracy:", accuracy, "\n")
```

Output:

```
> print(conf_matrix)
      Predicted
Actual  0  1
     0 42  6
     1 10 42
```

```
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.84
```

Step 4: Plotting the predicted probabilities versus actual probabilities



The data seem to fit well with the MARS model. Its accuracy is relatively high, and it is well-calibrated.

A small number of the data points are situated below the diagonal. This indicates that these students' actual chances of admission were higher than the model's predicted probability of admission.

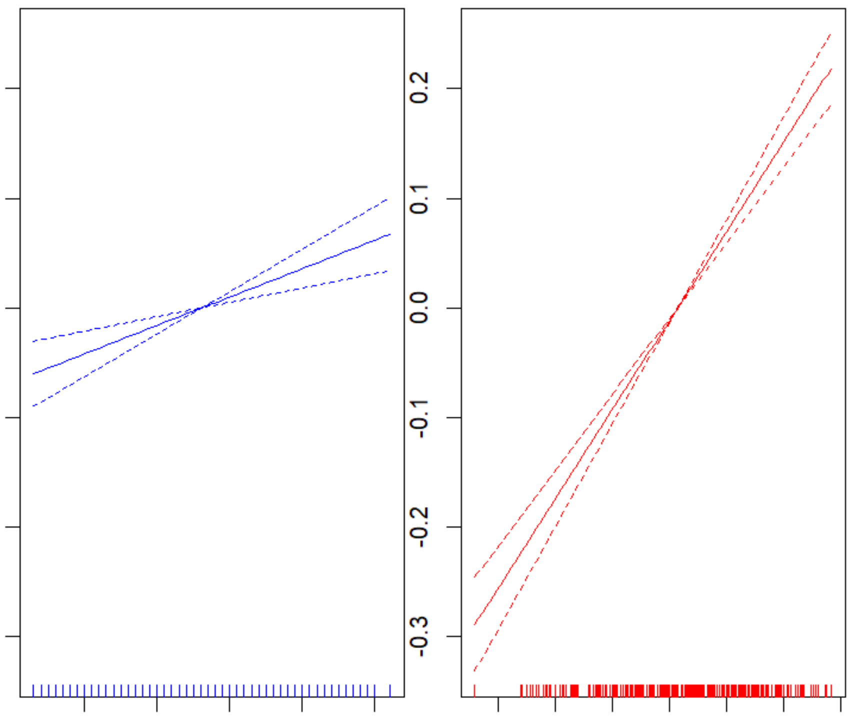
6. (20 points) Open question: Split the dataset into train and test. Fit a GAM on the training dataset and use the fitted model to predict the response variable on the test set. Report and interpret the results, and provide plots in order to discuss the performance of the model and its outputs.

Step 1: Fitting the GAM model and making predictions.

R code:

```r
# Fiting GAM model on training data
gam_model <- gam(Chance.of.Admit ~ s(GRE.Score) + s(CGPA), data = train_data)

# Predict on the test set
predicted_probabilities <- predict(gam_model, newdata = test_data)

par(mfrow = c(1, 2))
plot(gam_model)

names(plot_data) <- c("Actual_Chance", "Predicted_Probabilities", "Color")

```

Output:



Blue is the plot for GRE.Score and red graph is the plot for CGPA.

Step 2: Evaluating the model's performance with threshold 0.50.

R code:

```r
# Evaluate model performance
threshold <- 0.5
predicted_labels <- ifelse(predicted_probabilities > threshold, 1, 0)
actual_labels <- ifelse(test_data$Chance.of.Admit > 0.72, 1, 0)

# Confusion matrix
conf_matrix <- table(Actual = actual_labels, Predicted = predicted_labels)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
cat("Accuracy:", accuracy, "\n")

```

Output:

```
[1] "Confusion Matrix:"
> print(conf_matrix)
      Predicted
Actual  0  1
     0  6 42
     1  0 52
>
> # Calculate accuracy
> accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.58
> 
```

Step 3: Evaluating the model's performance with threshold 0.72.

R code:

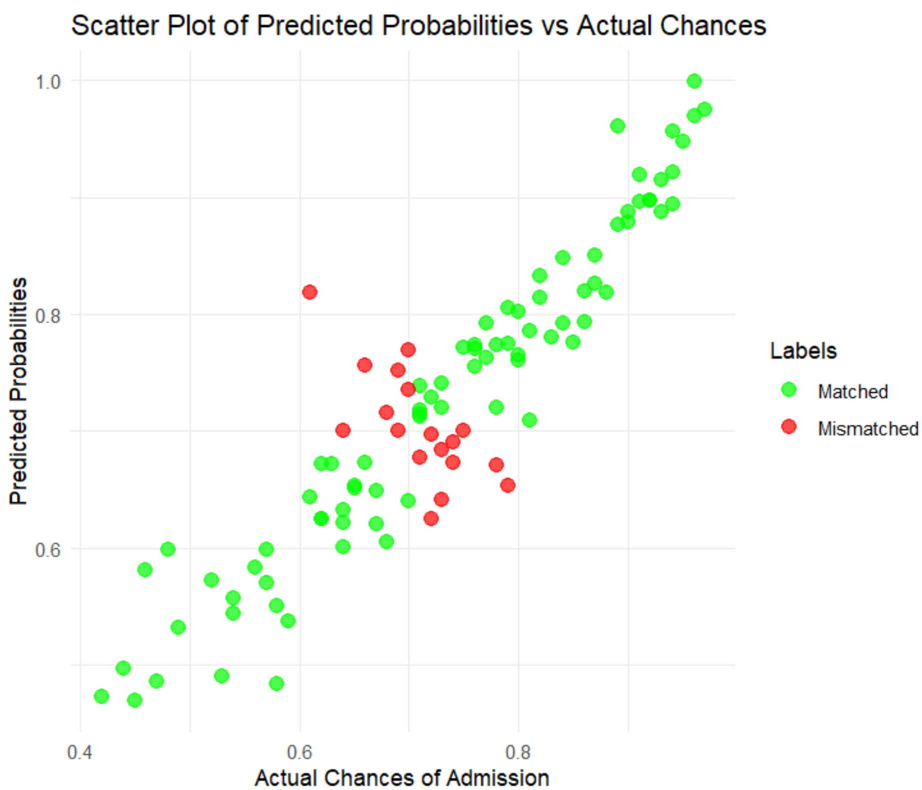
```r
threshold <- 0.72
predicted_labels <- ifelse(predicted_probabilities > threshold, 1, 0)
actual_labels <- ifelse(test_data$Chance.of.Admit > 0.72, 1, 0)

# Confusion matrix
conf_matrix <- table(Actual = actual_labels, Predicted = predicted_labels)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
cat("Accuracy:", accuracy, "\n")
```

Output:

```
> # Confusion matrix
> conf_matrix <- table(Actual = actual_labels, Predicted = predicted_labels)
> print("Confusion Matrix:")
[1] "Confusion Matrix:"
> print(conf_matrix)
      Predicted
Actual  0  1
     0 41  7
     1  9 43
>
> # Calculate accuracy
> accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.84
```

Step 4: Plotting the graph.

```
plot_data <- data.frame(
  Actual_Chance = test_data$Chance.of.Admit,
  Predicted_Probabilities = predicted_probabilities,
  Color = ifelse((predicted_probabilities > 0.7 & test_data$Chance.of.Admit <= 0.7) |
                   (predicted_probabilities <= 0.7 & test_data$Chance.of.Admit > 0.7), "red", "green")
)
# Plot the scatter plot with color-coded points
ggplot(plot_data, aes(x = Actual_Chance, y = Predicted_Probabilities, color = Color)) +
  geom_point(alpha = 0.7, size = 3) +
  ggtitle("Scatter Plot of Predicted Probabilities vs Actual Chances") +
  xlab("Actual Chances of Admission") +
  ylab("Predicted Probabilities") +
  scale_color_manual(values = c("green", "red"), name = "Labels",
                     labels = c("Matched", "Mismatched")) +
  theme_minimal()
```


Scatter Plot of Predicted Probabilities vs Actual Chances
1.0
0.8
0.6
Predicted Probabilities
0.4
0.6
0.8
Actual Chances of Admission
Labels
Matched
Mismatched