STAT527 Final Project

Edbert Jao

2024-05-28

Cleaning data, dealing with null values. For Glucose, BloodPressure, SkinThickness, Insulin, and BMI, it's biologically impossible to have a value of 0 so such values will be treated as NAs and be subject to mean imputation.

```
df <- read.csv("diabetes.csv");

paste("Number of observations of Glucose with a value of 0:", df %>% filter(Glucose == 0) %>%  nrow());
```

```
## [1] "Number of observations of Glucose with a value of 0: 5"
```

```
paste("Number of observations of BloodPressure with a value of 0:", df %>% filter(BloodPressure == 0) %
```

```
## [1] "Number of observations of BloodPressure with a value of 0: 35"
```

```
paste("Number of observations of SkinThickness with a value of 0:", df %>% filter(SkinThickness == 0) %
```

```
## [1] "Number of observations of SkinThickness with a value of 0: 227"
```

```
paste("Number of observations of Insulin with a value of 0:", df %>% filter(Insulin == 0) %>%  nrow());
```

```
## [1] "Number of observations of Insulin with a value of 0: 374"
```

```
paste("Number of observations of BMI with a value of 0:", df %>% filter(BMI == 0) %>%  nrow());
```

```
## [1] "Number of observations of BMI with a value of 0: 11"
```

```
# mean imputation
df <- df %>%
  mutate(Glucose = ifelse(Glucose == 0, NA, Glucose)) %>%
  mutate(Glucose = ifelse(is.na(Glucose), mean(Glucose, na.rm = TRUE), Glucose));
df <- df %>%
  mutate(BloodPressure = ifelse(BloodPressure == 0, NA, BloodPressure)) %>%
  mutate(BloodPressure = ifelse(is.na(BloodPressure), mean(BloodPressure, na.rm = TRUE), BloodPressure)
df <- df %>%
  mutate(SkinThickness = ifelse(SkinThickness == 0, NA, SkinThickness)) %>%
  mutate(SkinThickness = ifelse(is.na(SkinThickness), mean(SkinThickness, na.rm = TRUE), SkinThickness)
df <- df %>%
  mutate(Insulin = ifelse(Insulin == 0, NA, Insulin)) %>%
```

```r
  mutate(Insulin = ifelse(is.na(Insulin), mean(Insulin, na.rm = TRUE), Insulin));
df <- df %>%
  mutate(BMI = ifelse(BMI == 0, NA, BMI)) %>%
  mutate(BMI = ifelse(is.na(BMI), mean(BMI, na.rm = TRUE), BMI));
```

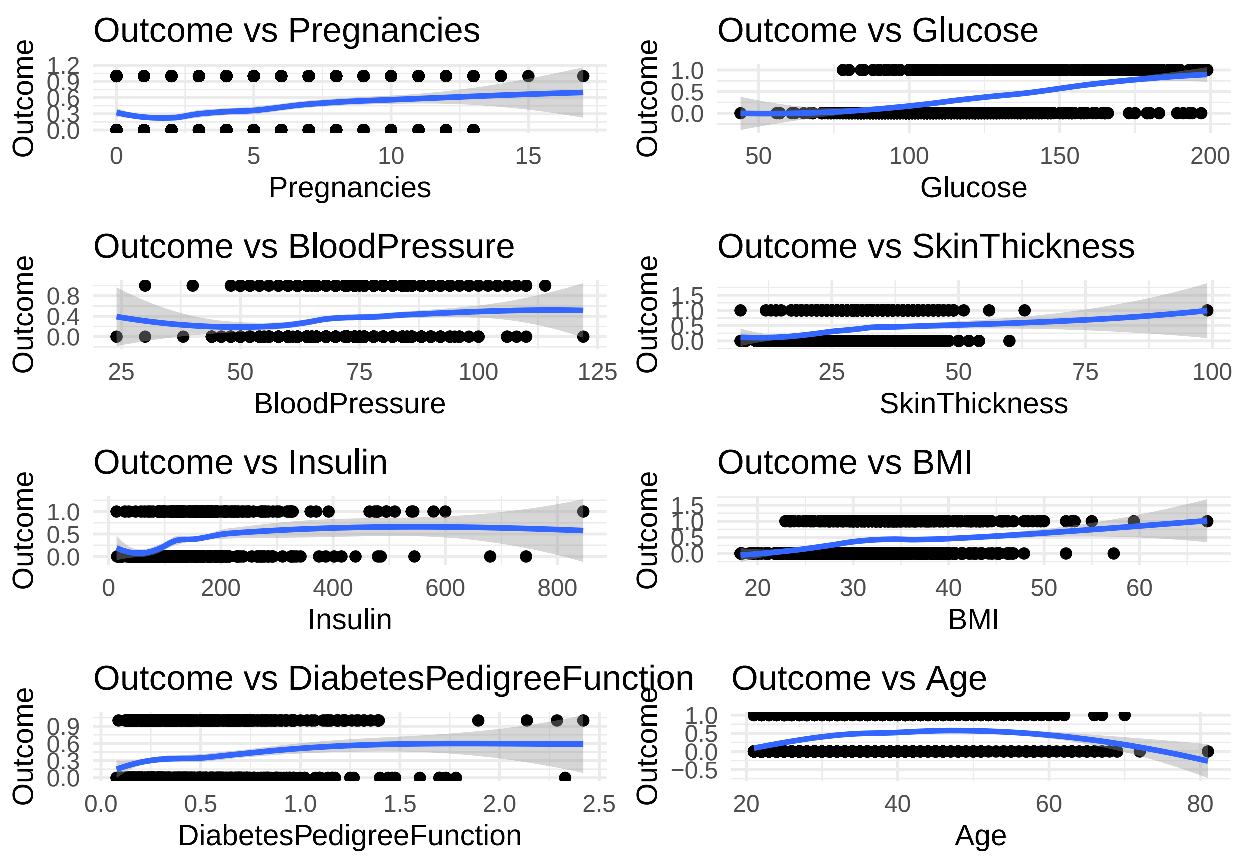
Here, R's built-in locally estimated scatterplot smoothing (i.e. local polynomial regression) is used to inspect the data and determine whether the explanatory variables have a nonlinear relationship with Outcome.

```r
columns <- c("Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPed

# Create the plots
plots <- lapply(columns, function(col) {
  ggplot(df, aes_string(x = col, y = "Outcome")) +
    geom_point() +
    geom_smooth(method = "loess") +
    labs(title = paste("Outcome vs", col), x = col, y = "Outcome") +
    theme_minimal()
});
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
grid.arrange(
  arrangeGrob(grobs = plots, ncol = 2,
              gp = gpar(fontsize = 20, fontface = "bold"))
);
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```



70:30 split of dataset into training and testing data:

```r
set.seed(527);

sample_indices <- sample(1:nrow(df), size = 0.7 * nrow(df))

# Create training and test sets
df_train <- df[ sample_indices, ];
df_test  <- df[-sample_indices, ];
```

Training models, using default R settings for the additive logistic models:

```r
formula_poly <- Outcome ~ poly(Pregnancies, 3) + poly(Glucose, 3) + poly(BloodPressure, 3) +
            poly(SkinThickness, 3) + poly(Insulin, 3) + poly(BMI, 3) +
            poly(DiabetesPedigreeFunction, 3) + poly(Age, 3);

formula_lo <- Outcome ~ lo(Pregnancies) + lo(Glucose) + lo(BloodPressure) +
            lo(SkinThickness) + lo(Insulin) + lo(BMI) +
            lo(DiabetesPedigreeFunction) + lo(Age);

formula_bs <- Outcome ~ bs(Pregnancies) + bs(Glucose) + bs(BloodPressure) +
            bs(SkinThickness) + bs(Insulin) + bs(BMI) +
            bs(DiabetesPedigreeFunction) + bs(Age);

formula_ns <- Outcome ~ ns(Pregnancies) + ns(Glucose) + ns(BloodPressure) +
            ns(SkinThickness) + ns(Insulin) + ns(BMI) +
            ns(DiabetesPedigreeFunction) + ns(Age);

# Fit the model with global polynomials
model_poly <- gam::gam(formula_poly, family = binomial(link = "logit"), data = df_train);
# Fit the model with local polynomials
model_lo <- gam::gam(formula_lo, family = binomial(link = "logit"), data = df_train);
# Fit the model with basis splines
model_bs <- gam::gam(formula_bs, family = binomial(link = "logit"), data = df_train);
# Fit the model with natural splines (additional constraint of linearity beyond boundary points)
model_ns <- gam::gam(formula_ns, family = binomial(link = "logit"), data = df_train);
```

Using models on Test Dataset and Visualizing the Results

```r
# Predict on the test data
predictions_poly <- predict(model_poly, newdata = df_test, type = "response");
predictions_lo   <- predict(model_lo,   newdata = df_test, type = "response");
```

```
## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : eval 44

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : lowerlimit 61.32

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : eval 57

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : lowerlimit 61.32

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : eval 61

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : lowerlimit 61.32

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : eval 57

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : lowerlimit 61.32
## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : eval 199

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : upperlimit 198.68

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : eval 56

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : lowerlimit 61.32

## Warning in gam.lo(data[["lo(Glucose)"]], z, w, span = 0.5, degree = 1, ncols =
## 1, : extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(BloodPressure)"]], z, w, span = 0.5, degree = 1, :
## eval 122

## Warning in gam.lo(data[["lo(BloodPressure)"]], z, w, span = 0.5, degree = 1, :
## upperlimit 114.42

## Warning in gam.lo(data[["lo(BloodPressure)"]], z, w, span = 0.5, degree = 1, :
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(BloodPressure)"]], z, w, span = 0.5, degree = 1, :
## eval 24

## Warning in gam.lo(data[["lo(BloodPressure)"]], z, w, span = 0.5, degree = 1, :
## lowerlimit 29.58

## Warning in gam.lo(data[["lo(BloodPressure)"]], z, w, span = 0.5, degree = 1, :
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(BMI)"]], z, w, span = 0.5, degree = 1, ncols = 1, :
## eval 67.1

## Warning in gam.lo(data[["lo(BMI)"]], z, w, span = 0.5, degree = 1, ncols = 1, :
## upperlimit 59.606

## Warning in gam.lo(data[["lo(BMI)"]], z, w, span = 0.5, degree = 1, ncols = 1, :
## extrapolation not allowed with blending
```

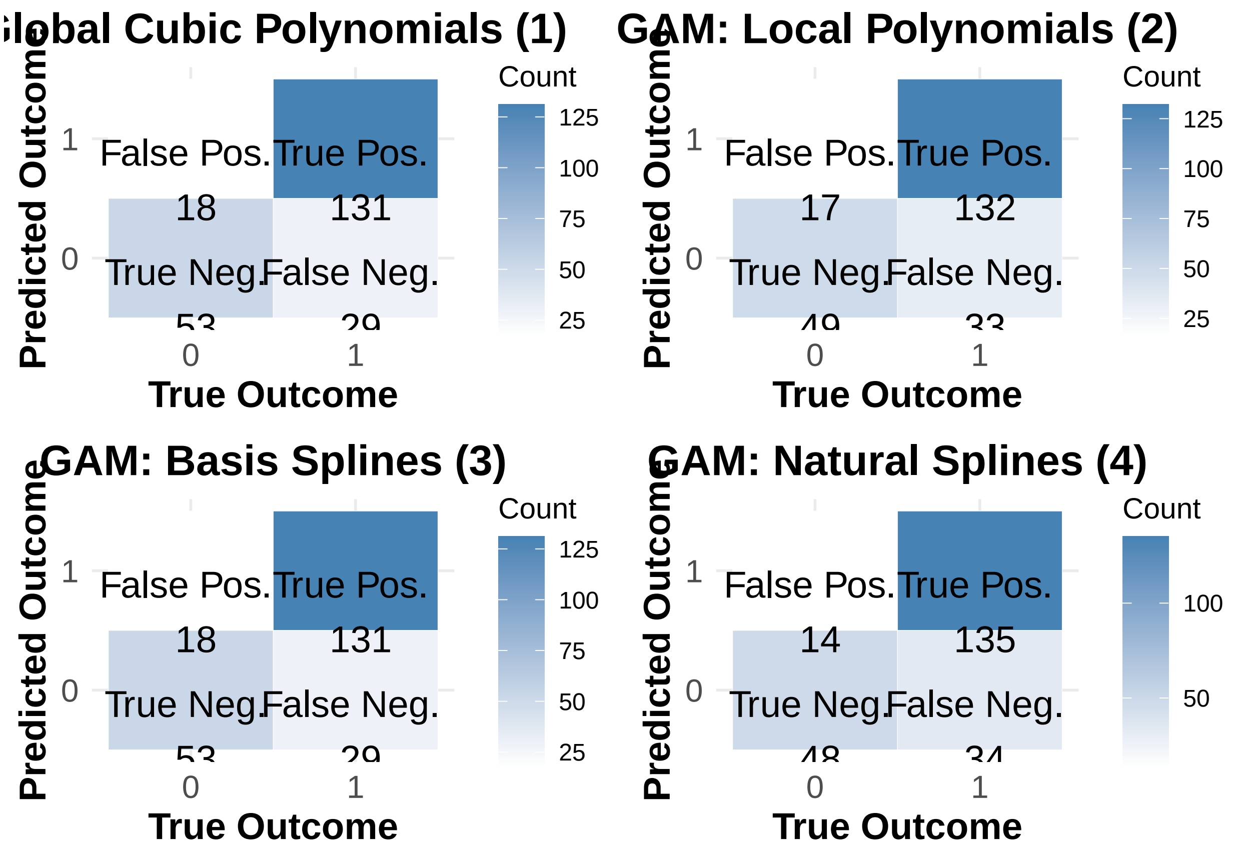
```r
predictions_bs   <- predict(model_bs,   newdata = df_test, type = "response");
```

```
## Warning in bs(Glucose, degree = 3L, knots = numeric(0), Boundary.knots = c(62,
## : some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(BloodPressure, degree = 3L, knots = numeric(0), Boundary.knots =
## c(30, : some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(BMI, degree = 3L, knots = numeric(0), Boundary.knots = c(18.2, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```r
predictions_ns   <- predict(model_ns,   newdata = df_test, type = "response");

# Combine predictions and actual data into a single data frame
df_test_predictions <- df_test %>%
  mutate(Prediction_Poly = predictions_poly,
         Prediction_LO = predictions_lo,
         Prediction_BS = predictions_bs,
         Prediction_NS = predictions_ns,
         Actual = Outcome);

pred_class_poly <- ifelse(predictions_poly > 0.5, 1, 0);
pred_class_lo   <- ifelse(predictions_lo > 0.5, 1, 0);
pred_class_bs   <- ifelse(predictions_bs > 0.5, 1, 0);
pred_class_ns   <- ifelse(predictions_ns > 0.5, 1, 0);

# Create confusion matrices
conf_matrix_poly <- confusionMatrix(factor(pred_class_poly), factor(df_test$Outcome));
conf_matrix_lo   <- confusionMatrix(factor(pred_class_lo), factor(df_test$Outcome));
conf_matrix_bs   <- confusionMatrix(factor(pred_class_bs), factor(df_test$Outcome));
conf_matrix_ns   <- confusionMatrix(factor(pred_class_ns), factor(df_test$Outcome));

# Function to plot confusion matrix
plot_confusion_matrix <- function(conf_matrix, title) {
  conf_df <- as.data.frame(conf_matrix$table);
  colnames(conf_df) <- c("Reference", "Prediction", "Frequency");

  # Reorder the factors to match the desired layout
  conf_df$Prediction <- factor(conf_df$Prediction, levels = c(1, 0));
  conf_df$Reference <- factor(conf_df$Reference, levels = c(1, 0));

  # Add labels for TP, FP, FN, TN
  conf_df$Label <- c("True Pos.", "False Pos.", "False Neg.", "True Neg.");

  ggplot(conf_df, aes(x = Reference, y = Prediction)) +
    geom_tile(aes(fill = Frequency), color = "white") +
    scale_fill_gradient(low = "white", high = "steelblue") +
    geom_text(aes(label = paste(Label, "\n", Frequency)), vjust = 1, size = 5) +
    labs(title = title, fill = "Count") +
    theme_minimal() +
    theme(
      axis.title.x = element_text(size = 14, face = "bold"),
      axis.title.y = element_text(size = 14, face = "bold"),
      plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
      axis.text = element_text(size = 12)
    ) +
    scale_x_discrete(labels = c("0", "1")) +
    scale_y_discrete(labels = c("0", "1")) +
    labs(x = "True Outcome", y = "Predicted Outcome");
};

# Create plots of confusion matrices
plot1 <- plot_confusion_matrix(conf_matrix_poly, "Global Cubic Polynomials (1)");
plot2 <- plot_confusion_matrix(conf_matrix_lo, "GAM: Local Polynomials (2)");
plot3 <- plot_confusion_matrix(conf_matrix_bs, "GAM: Basis Splines (3)");
plot4 <- plot_confusion_matrix(conf_matrix_ns, "GAM: Natural Splines (4)");

# Display the plots
print(grid.arrange(plot1, plot2, plot3, plot4, ncol = 2, nrow = 2));
```



```
## TableGrob (2 x 2) "arrange": 4 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
## 3 3 (2-2,1-1) arrange gtable[layout]
## 4 4 (2-2,2-2) arrange gtable[layout]
```

```
roc_poly <- roc(df_test$Outcome, predictions_poly);
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
roc_lo   <- roc(df_test$Outcome,   predictions_lo);
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc_bs   <- roc(df_test$Outcome,   predictions_bs);
```

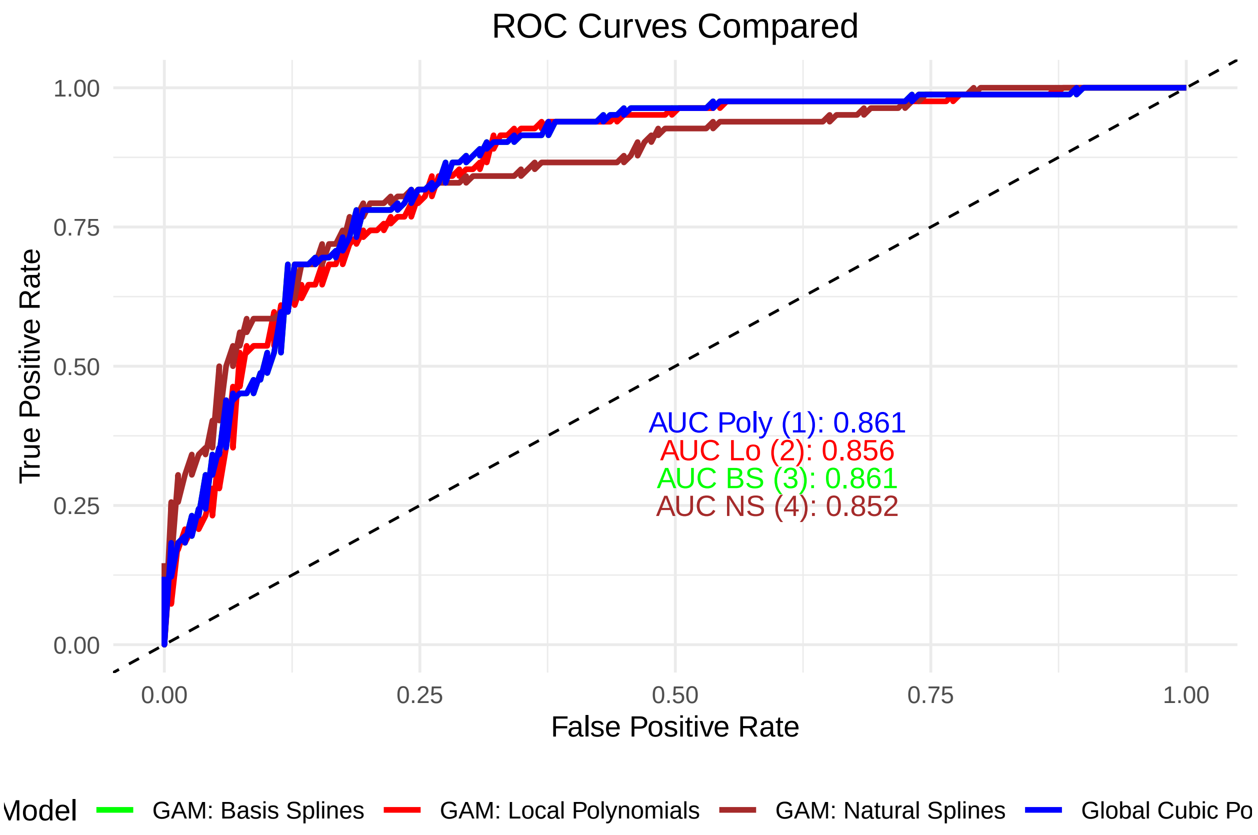
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc_ns   <- roc(df_test$Outcome,   predictions_ns);
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
# Convert ROC objects to data frames
roc_poly_df <- data.frame(tpr = roc_poly$sensitivities, fpr = 1 - roc_poly$specificities, model = "Globa
roc_lo_df   <- data.frame(tpr = roc_lo$sensitivities,   fpr = 1 - roc_lo$specificities, model = "GAM: Lo
roc_bs_df   <- data.frame(tpr = roc_bs$sensitivities,   fpr = 1 - roc_bs$specificities, model = "GAM: Ba
roc_ns_df   <- data.frame(tpr = roc_ns$sensitivities,   fpr = 1 - roc_ns$specificities, model = "GAM: Na


# Combine data frames
roc_combined_df <- bind_rows(roc_poly_df, roc_lo_df, roc_bs_df, roc_ns_df);


# Manually define model colors
model_colors <- c("Global Cubic Polynomials" = "blue",
                  "GAM: Local Polynomials" = "red",
                  "GAM: Basis Splines" = "green",
                  "GAM: Natural Splines" = "brown");

# Plot using ggplot2
ggplot(roc_combined_df, aes(x = fpr, y = tpr, color = model)) +
  geom_line(size = 1) +
  scale_color_manual(values = model_colors) +
  geom_abline(linetype = "dashed") +
  labs(title = "ROC Curves Compared",
       x = "False Positive Rate",
       y = "True Positive Rate",
       color = "Model") +
  theme_minimal() +
  theme(legend.position = "bottom", plot.title = element_text(hjust = 0.5)) +
  annotate("text", x = 0.6, y = 0.4, label = paste("AUC Poly (1):", round(roc_poly$auc, 3)), color = "bl
  annotate("text", x = 0.6, y = 0.35, label = paste("AUC Lo (2):", round(roc_lo$auc, 3)), color = "red")
  annotate("text", x = 0.6, y = 0.3, label = paste("AUC BS (3):", round(roc_bs$auc, 3)), color = "green
  annotate("text", x = 0.6, y = 0.25, label = paste("AUC NS (4):", round(roc_ns$auc, 3)), color = "brow
```

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



```
# Function to calculate log-loss
log_loss <- function(actual, predicted) {
  # Clip predicted values to avoid log(0) issues
  predicted <- pmax(pmin(predicted, 1 - 1e-15), 1e-15);
  -mean(actual * log(predicted) + (1 - actual) * log(1 - predicted));
};

# Calculate log-loss for each model
log_loss_poly <- log_loss(df_test$Outcome, predictions_poly);
log_loss_lo   <- log_loss(df_test$Outcome, predictions_lo);
log_loss_bs   <- log_loss(df_test$Outcome, predictions_bs);
log_loss_ns   <- log_loss(df_test$Outcome, predictions_ns);

# Display log-loss values
```

```
log_loss_results <- data.frame(
  Model = c("Global Cubic Polynomials (1)", "GAM: Local Polynomials (2)", "GAM: Basis Splines (3)", "GAI
  LogLoss = c(log_loss_poly, log_loss_lo, log_loss_bs, log_loss_ns));

print(log_loss_results);
```

```
##                          Model   LogLoss
## 1 Global Cubic Polynomials (1) 0.4481480
## 2   GAM: Local Polynomials (2) 0.4560115
## 3       GAM: Basis Splines (3) 0.4481480
## 4     GAM: Natural Splines (4) 0.4520909
```