Data 624 - HW8 (Fall 2024)

Khyati Naik

7.2

Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data: y = 10 sin(x1x2) + 20(x3 − 0.5)2 + 10x4 + 5x5 + N(0, 2) where the x values are random variables uniformly distributed between [0, 1] (there are also 5 other non-informative variables also created in the simulation). The package mlbench contains a function called mlbench.friedman1 that simulates these data. Tune several models on these data.Which models appear to give the best performance? Does MARS select the informative predictors (those named X1–X5)?

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(nnet)
library(earth)
```

```
## Loading required package: Formula
```

```
## Loading required package: plotmo
```

```
## Loading required package: plotrix
```

```
library(kernlab)
```

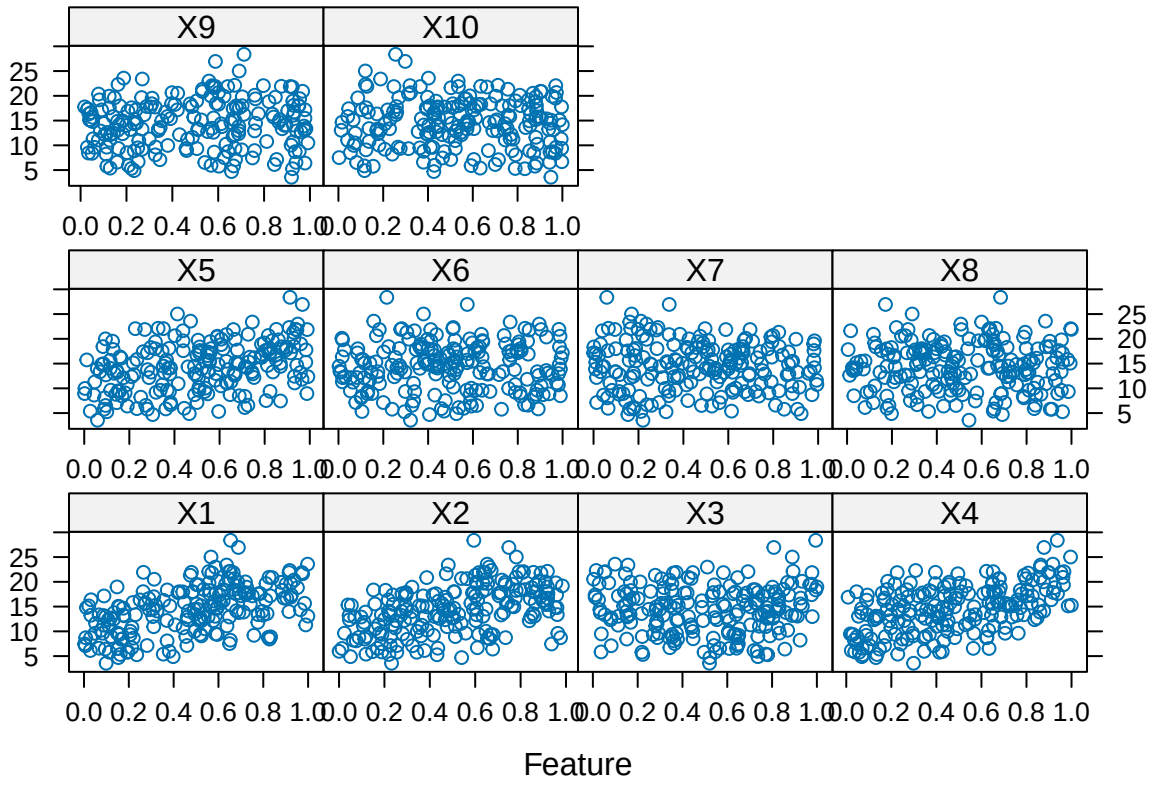
```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library(mlbench)
library(AppliedPredictiveModeling)
library(RANN)
library(dplyr)
```

```
##
## 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
```

```r
set.seed(200)
trainingData <- mlbench.friedman1(200, sd = 1)
## We convert the 'x" data from a matrix to a dataframe
## One reason is that this will give the columns names.
trainingData$x <- data.frame(trainingData$x)
## Look at the data using
featurePlot(trainingData$x, trainingData$y)
```




```r
## or other methods.

## This creates a list with a vector 'y' and a maxtrix
## of predictors 'x'. Also simulates a large test set to
## estimate the true error rate with good precision:
```

```r
testData <- mlbench.friedman1(5000, sd = 1)
testData$x <- data.frame(testData$x)

## Example of Tuning a Model is
knnmodel <- train(x = trainingData$x,
                  y = trainingData$y,
                  method = "knn",
                  preProc = c("center", "scale"),
                  tuneLength = 10)
knnmodel
```

```
## k-Nearest Neighbors
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  3.466085  0.5121775  2.816838
##    7  3.349428  0.5452823  2.727410
##    9  3.264276  0.5785990  2.660026
##   11  3.214216  0.6024244  2.603767
##   13  3.196510  0.6176570  2.591935
##   15  3.184173  0.6305506  2.577482
##   17  3.183130  0.6425367  2.567787
##   19  3.198752  0.6483184  2.592683
##   21  3.188993  0.6611428  2.588787
##   23  3.200458  0.6638353  2.604529
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 17.
```

```r
knnpred <- predict(knnmodel, newdata = testData$x)
## the function 'postResample' can be used to get the test set
## performance values
postResample(pred = knnpred, obs = testData$y)
```

```
##      RMSE  Rsquared       MAE
## 3.2040595 0.6819919 2.5683461
```

```r
# Single-Layer Neural Network with 5 Hidden Neurons

# Train a single-layer neural network model
single_layer_nn <- nnet(
  x = trainingData$x,               # Input features from training dataset
  y = trainingData$y,               # Target output values from training dataset
  size = 5,                         # Number of neurons in the hidden layer
  decay = 0.01,                     # Regularization parameter to reduce overfitting
```

```
  linout = TRUE,                    # Use linear output for regression
  trace = FALSE,                    # Suppress training output details
  maxit = 500,                      # Maximum number of iterations for convergence
  MaxNWts = 5 * (ncol(trainingData$x) + 1) + 5 + 1 # Max weights based on inputs and hidden layer size
)

# Predict using the single-layer model on test dataset
single_layer_predictions <- predict(single_layer_nn, newdata = testData$x)

# Evaluate model performance: accuracy, RMSE, and R-squared
single_layer_performance <- postResample(pred = single_layer_predictions, obs = testData$y)

# Averaged Neural Network Model with Multiple Repeats

# Train an ensemble neural network model with averaged predictions
averaged_nn_model <- avNNet(
  x = trainingData$x,               # Input features from training dataset
  y = trainingData$y,               # Target output values from training dataset
  size = 5,                         # Number of neurons in each hidden layer
  decay = 0.01,                     # Regularization parameter to reduce overfitting
  repeats = 5,                      # Number of model repeats for ensemble averaging
  linout = TRUE,                    # Use linear output for regression
  trace = FALSE,                    # Suppress training output details
  maxit = 500,                      # Maximum number of iterations for convergence
  MaxNWts = 5 * (ncol(trainingData$x) + 1) + 5 + 1 # Max weights based on inputs and hidden layer size
)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
# Predict using the averaged model on test dataset
averaged_nn_predictions <- predict(averaged_nn_model, newdata = testData$x)

# Evaluate performance of the averaged neural network model
averaged_nn_performance <- postResample(pred = averaged_nn_predictions, obs = testData$y)

# Output the performance metrics for both models
list(Single_Layer_NN_Performance = single_layer_performance,
     Averaged_NN_Performance = averaged_nn_performance)
```

```
## $Single_Layer_NN_Performance
##      RMSE  Rsquared       MAE
## 2.5244877 0.7621849 1.8722728
##
## $Averaged_NN_Performance
##      RMSE  Rsquared       MAE
## 2.0404770 0.8331947 1.5329094
```

```
# Regular MARS (Multivariate Adaptive Regression Splines) Model

# Train a basic MARS model on the training data
basic_mars_model <- earth(
  x = trainingData$x,               # Input features from training dataset
```

```r
  y = trainingData$y                # Target output values from training dataset
)

# Display model summary for interpretation
print(basic_mars_model)
```

```
## Selected 12 of 18 terms, and 6 of 10 predictors
## Termination condition: Reached nk 21
## Importance: X1, X4, X2, X5, X3, X6, X7-unused, X8-unused, X9-unused, ...
## Number of terms at each degree of interaction: 1 11 (additive model)
## GCV 2.540556    RSS 397.9654    GRSq 0.8968524    RSq 0.9183982
```

```r
summary(basic_mars_model)
```

```
## Call: earth(x=trainingData$x, y=trainingData$y)
##
##                coefficients
## (Intercept)       18.451984
## h(0.621722-X1)   -11.074396
## h(0.601063-X2)   -10.744225
## h(X3-0.281766)    20.607853
## h(0.447442-X3)    17.880232
## h(X3-0.447442)   -23.282007
## h(X3-0.636458)    15.150350
## h(0.734892-X4)   -10.027487
## h(X4-0.734892)     9.092045
## h(0.850094-X5)    -4.723407
## h(X5-0.850094)    10.832932
## h(X6-0.361791)    -1.956821
##
## Selected 12 of 18 terms, and 6 of 10 predictors
## Termination condition: Reached nk 21
## Importance: X1, X4, X2, X5, X3, X6, X7-unused, X8-unused, X9-unused, ...
## Number of terms at each degree of interaction: 1 11 (additive model)
## GCV 2.540556    RSS 397.9654    GRSq 0.8968524    RSq 0.9183982
```

```r
# Make predictions with the basic MARS model on the test dataset
basic_mars_predictions <- predict(basic_mars_model, newdata = testData$x)

# Evaluate performance of the basic MARS model: accuracy, RMSE, and R-squared
basic_mars_performance <- postResample(pred = basic_mars_predictions, obs = testData$y)

# Setting up a grid for hyperparameter tuning
mars_tuning_grid <- expand.grid(.degree = 1:2, .nprune = 2:16)  # Degrees of interaction and number of

# Set seed for reproducibility in cross-validation
set.seed(1234)

# Tuning the MARS model with cross-validation

# Train a MARS model with cross-validation to find optimal hyperparameters
tuned_mars_model <- train(
```

```r
  x = trainingData$x,                # Input features from training dataset
  y = trainingData$y,                # Target output values from training dataset
  method = "earth",                  # Specifies MARS model
  tuneGrid = mars_tuning_grid,       # Grid of tuning parameters
  trControl = trainControl(method = "cv") # Cross-validation control
)

# Display the tuned model details
print(tuned_mars_model)
```

```
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      4.312373  0.2464574  3.547735
##   1        3      3.707278  0.4683798  3.005934
##   1        4      2.663179  0.7126715  2.137120
##   1        5      2.261502  0.7904454  1.824339
##   1        6      2.145367  0.8119436  1.680939
##   1        7      1.797464  0.8635322  1.419835
##   1        8      1.720995  0.8768729  1.389307
##   1        9      1.653852  0.8890984  1.326900
##   1       10      1.674588  0.8827378  1.320200
##   1       11      1.650636  0.8864692  1.284660
##   1       12      1.646699  0.8855752  1.288449
##   1       13      1.643453  0.8866051  1.288624
##   1       14      1.646405  0.8856334  1.291766
##   1       15      1.637579  0.8876260  1.283147
##   1       16      1.637579  0.8876260  1.283147
##   2        2      4.312373  0.2464574  3.547735
##   2        3      3.721908  0.4678488  3.016069
##   2        4      2.674839  0.7105853  2.140857
##   2        5      2.267824  0.7888276  1.832740
##   2        6      2.216987  0.8005498  1.778172
##   2        7      1.844249  0.8605997  1.431686
##   2        8      1.749633  0.8774372  1.360255
##   2        9      1.560708  0.9024603  1.231461
##   2       10      1.529405  0.9066803  1.222445
##   2       11      1.467736  0.9105211  1.153231
##   2       12      1.360016  0.9236123  1.084557
##   2       13      1.304167  0.9332150  1.048825
##   2       14      1.378462  0.9261006  1.101009
##   2       15      1.395984  0.9244996  1.114657
##   2       16      1.424798  0.9209343  1.131557
##
## RMSE was used to select the optimal model using the smallest value.
```

```
## The final values used for the model were nprune = 13 and degree = 2.
```

```r
# Make predictions with the tuned MARS model on the test dataset
tuned_mars_predictions <- predict(tuned_mars_model, newdata = testData$x)

# Evaluate performance of the tuned MARS model
tuned_mars_performance <- postResample(pred = tuned_mars_predictions, obs = testData$y)

# Output performance metrics for both the basic and tuned MARS models
list(Basic_MARS_Performance = basic_mars_performance,
     Tuned_MARS_Performance = tuned_mars_performance)
```

```
## $Basic_MARS_Performance
##      RMSE  Rsquared       MAE
## 1.8136467 0.8677298 1.3911836
##
## $Tuned_MARS_Performance
##      RMSE  Rsquared       MAE
## 1.2803060 0.9335241 1.0168673
```

```r
# Training SVM with Radial Basis Function (RBF) Kernel

# Train an SVM model with an RBF kernel on the training data
svm_rbf_model <- ksvm(
  x = as.matrix(trainingData$x),     # Input features matrix from training dataset
  y = trainingData$y,                # Target output values from training dataset
  kernel = "rbfdot",                 # Use radial basis function kernel
  kpar = "automatic",                # Automatically select kernel parameters
  C = 1,                             # Regularization parameter
  epsilon = 0.1                      # Epsilon parameter for regression tolerance
)

# Make predictions with the RBF SVM model on the test dataset
svm_rbf_predictions <- predict(svm_rbf_model, newdata = testData$x)

# Evaluate performance of the RBF SVM model: accuracy, RMSE, and R-squared
svm_rbf_performance <- postResample(pred = svm_rbf_predictions, obs = testData$y)

# Tuning SVM with Radial Kernel, Centering, and Scaling

# Set up an SVM model with cross-validation, centering, and scaling for tuning
tuned_svm_model <- train(
  x = trainingData$x,                # Input features from training dataset
  y = trainingData$y,                # Target output values from training dataset
  method = "svmRadial",              # SVM with radial basis kernel
  preProc = c("center", "scale"),    # Preprocess data by centering and scaling
  tuneLength = 14,                   # Number of hyperparameter combinations to try
  trControl = trainControl(method = "cv") # Cross-validation control
)

# Display the details of the final tuned model
print(tuned_svm_model$finalModel)
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: eps-svr  (regression)
##  parameter : epsilon = 0.1  cost C = 8
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0602551759408374
##
## Number of Support Vectors : 153
##
## Objective Function Value : -77.0787
## Training error : 0.009431
```

```
# Make predictions with the tuned SVM model on the test dataset
tuned_svm_predictions <- predict(tuned_svm_model, newdata = testData$x)

# Evaluate performance of the tuned SVM model
tuned_svm_performance <- postResample(pred = tuned_svm_predictions, obs = testData$y)

# Output performance metrics for both the basic and tuned SVM models
list(RBF_SVM_Performance = svm_rbf_performance,
     Tuned_SVM_Performance = tuned_svm_performance)
```

```
## $RBF_SVM_Performance
##      RMSE  Rsquared       MAE
## 2.2550806 0.7996157 1.7224336
##
## $Tuned_SVM_Performance
##      RMSE  Rsquared       MAE
## 2.0437049 0.8307058 1.5500663
```

In comparing the base models of four different non-linear methods—Multivariate Adaptive Regression Splines (MARS), Support Vector Machine (SVM), Neural Network (NN), and K-Nearest Neighbor (KNN)—we observe that MARS achieves the strongest baseline performance, with an RMSE of 1.8 and an R-squared of 0.86. This indicates that MARS captures the data structure better than the other models at this base level.

Looking at the tuned models, MARS again outperforms the others with an RMSE of 1.28 and an R-squared of 0.93, confirming its adaptability in capturing non-linear relationships effectively even after hyperparameter tuning. Among the remaining three models, the ranking changes slightly: for base models, SVM shows better performance than NN, followed by KNN. However, after tuning, the Neural Network slightly surpasses the SVM, suggesting that NN benefits more from tuning adjustments than SVM does in this case.

In terms of feature selection, MARS naturally identifies and prioritizes informative predictors (x1-x5) while leaving less relevant features (x7-x10) unused. This inherent ability to select the most predictive variables is advantageous, as it not only improves model accuracy but also provides interpretability. The five top-ranked predictors—x1 through x5—stand out with high importance scores, while x6 is slightly less significant, and the remaining predictors contribute minimally to model performance.

7.5

Exercise 6.3 describes data for a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several nonlinear regression models.

(a) Which nonlinear regression model gives the optimal resampling and test set performance?

```
# Load dataset
data("ChemicalManufacturingProcess")

# Step 1: Handle Missing Values Using KNN Imputation
# The 'preProcess' function is set up to fill in missing data using KNN imputation.
imputation_model <- preProcess(ChemicalManufacturingProcess, method = "knnImpute")
data_imputed <- predict(imputation_model, ChemicalManufacturingProcess)

# Separate predictors (X) and response variable (y)
predictors <- dplyr::select(data_imputed, -Yield)  # All variables except 'Yield'
response <- data_imputed$Yield                     # 'Yield' is the target variable

# Part (a): Train-Test Split for Model Evaluation

# Set random seed for reproducibility and create an 80-20 train-test split
set.seed(1234)
train_indices <- createDataPartition(response, p = 0.8, list = FALSE)
train_X <- predictors[train_indices, ] %>% as.matrix()  # Training predictors matrix
test_X <- predictors[-train_indices, ] %>% as.matrix()   # Testing predictors matrix
train_y <- response[train_indices]                      # Training target values
test_y <- response[-train_indices]                      # Testing target values

# 1. K-Nearest Neighbors (KNN) Model
# This model scales and centers the data, tuning for the best K value within a range of 10.
knn_model <- train(
  x = train_X, y = train_y,
  method = "knn",
  preProc = c("center", "scale"),
  tuneLength = 10
)
```

```
## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07
```

```
## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: BiologicalMaterial07
```

```
knn_predictions <- predict(knn_model, newdata = test_X)
postResample(pred = knn_predictions, obs = test_y)
```

```
##      RMSE  Rsquared       MAE
## 0.7059851 0.3434676 0.5682675
```

```
# Output: Performance metrics (e.g., RMSE, R-squared) to assess KNN model's accuracy.

# 2. Multivariate Adaptive Regression Splines (MARS) Model with Cross-Validation Tuning
# Expanding grid for tuning parameters degree and nprune for the MARS model.
mars_params <- expand.grid(.degree = 1:2, .nprune = 2:58)
mars_tuned_model <- train(
  x = train_X, y = train_y,
  method = "earth",
  tuneGrid = mars_params,
  trControl = trainControl(method = "cv")
)
mars_predictions <- predict(mars_tuned_model, newdata = test_X)
postResample(pred = mars_predictions, obs = test_y)
```

```
##      RMSE  Rsquared       MAE
## 0.5891524 0.5427526 0.4669305
```

```
# Output: Performance metrics for MARS model on test data.

# 3. Support Vector Machine (SVM) with Radial Basis Function Kernel
# Tuning SVM model using cross-validation with radial kernel, centering, and scaling.
svm_model <- train(
  x = train_X, y = train_y,
  method = "svmRadial",
  preProc = c("center", "scale"),
  tuneLength = 14,
  trControl = trainControl(method = "cv")
)
svm_predictions <- predict(svm_model, newdata = test_X)
postResample(pred = svm_predictions, obs = test_y)
```

```
##      RMSE  Rsquared       MAE
## 0.6095296 0.5236468 0.4804063
```

```
# Output: Accuracy metrics for tuned SVM model.

# 4. Neural Network Model
# Training an averaged neural network with 5 hidden nodes and weight decay.
nn_model <- avNNet(
  x = train_X, y = train_y,
  size = 5,
  decay = 0.01,
  repeats = 5,
  linout = TRUE,
  trace = FALSE,
  maxit = 500,
  MaxNWts = 5 * (ncol(train_X) + 1) + 5 + 1
)
nn_predictions <- predict(nn_model, newdata = test_X)
postResample(pred = nn_predictions, obs = test_y)
```

```
##      RMSE  Rsquared       MAE
## 0.7016608 0.4516523 0.5697754
```

```
# Output: Performance metrics for Neural Network model.
```

(b) Which predictors are most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?

```
# Part (b): Variable Importance from MARS Model
# Variable importance to determine which features are most informative.
importance_mars <- varImp(mars_tuned_model)
print(importance_mars)
```

```
## earth variable importance
##
##                         Overall
## ManufacturingProcess32 100.000
## ManufacturingProcess09  59.823
## ManufacturingProcess13  17.294
## ManufacturingProcess01   2.199
## ManufacturingProcess42   0.000
```

```
# Fitting MARS model directly for comparison of feature importance
mars_direct <- earth(x = train_X, y = train_y)
importance_direct <- varImp(mars_direct)
print(importance_direct)
```

```
##                          Overall
## ManufacturingProcess32 100.00000
```

```
## ManufacturingProcess09  67.41221
## ManufacturingProcess13  35.46817
## ManufacturingProcess01  26.58793
## ManufacturingProcess42  25.56440
## ManufacturingProcess33  21.70760
## BiologicalMaterial03    15.11625
## ManufacturingProcess22  13.16129
## ManufacturingProcess28  10.32610
## ManufacturingProcess04   0.00000
```

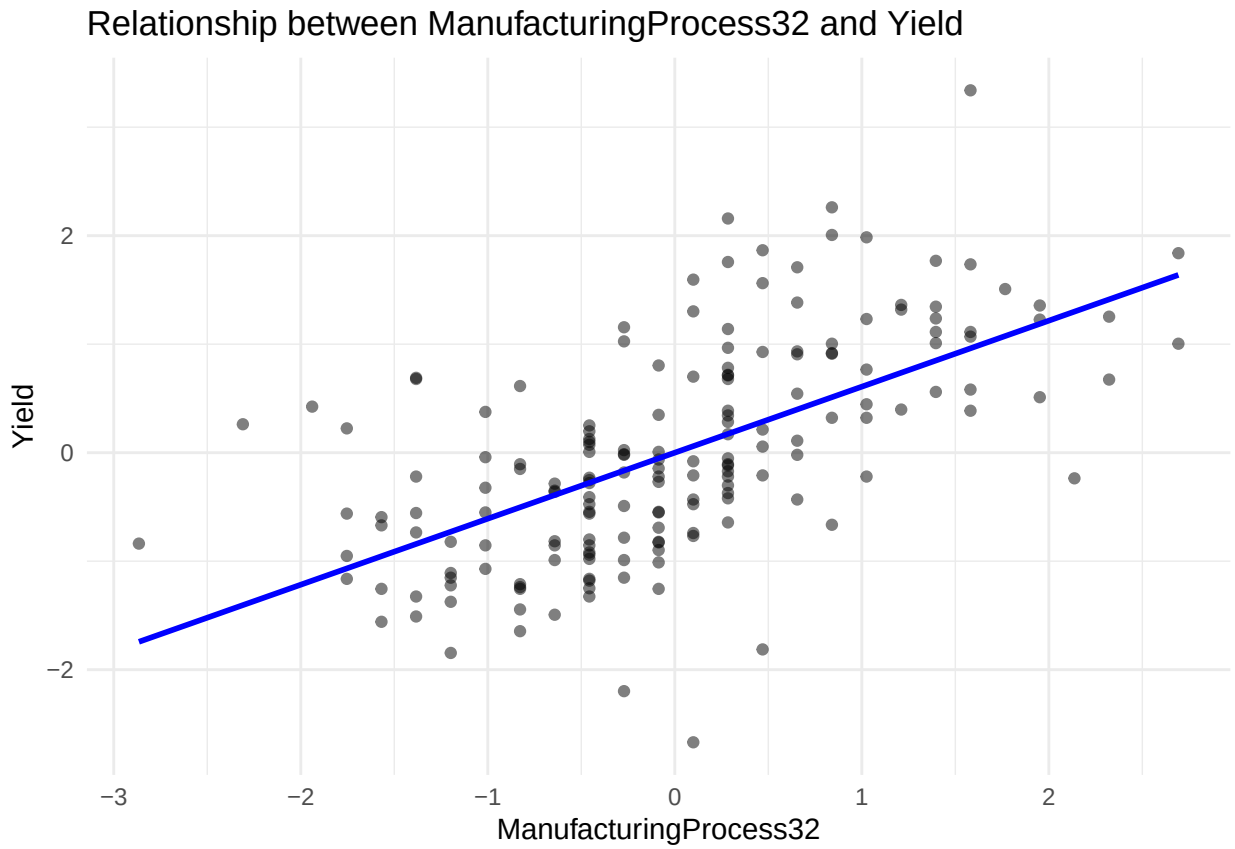
```
# Output: Displays variable importance, showing top predictors used by MARS.
```

(c) Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

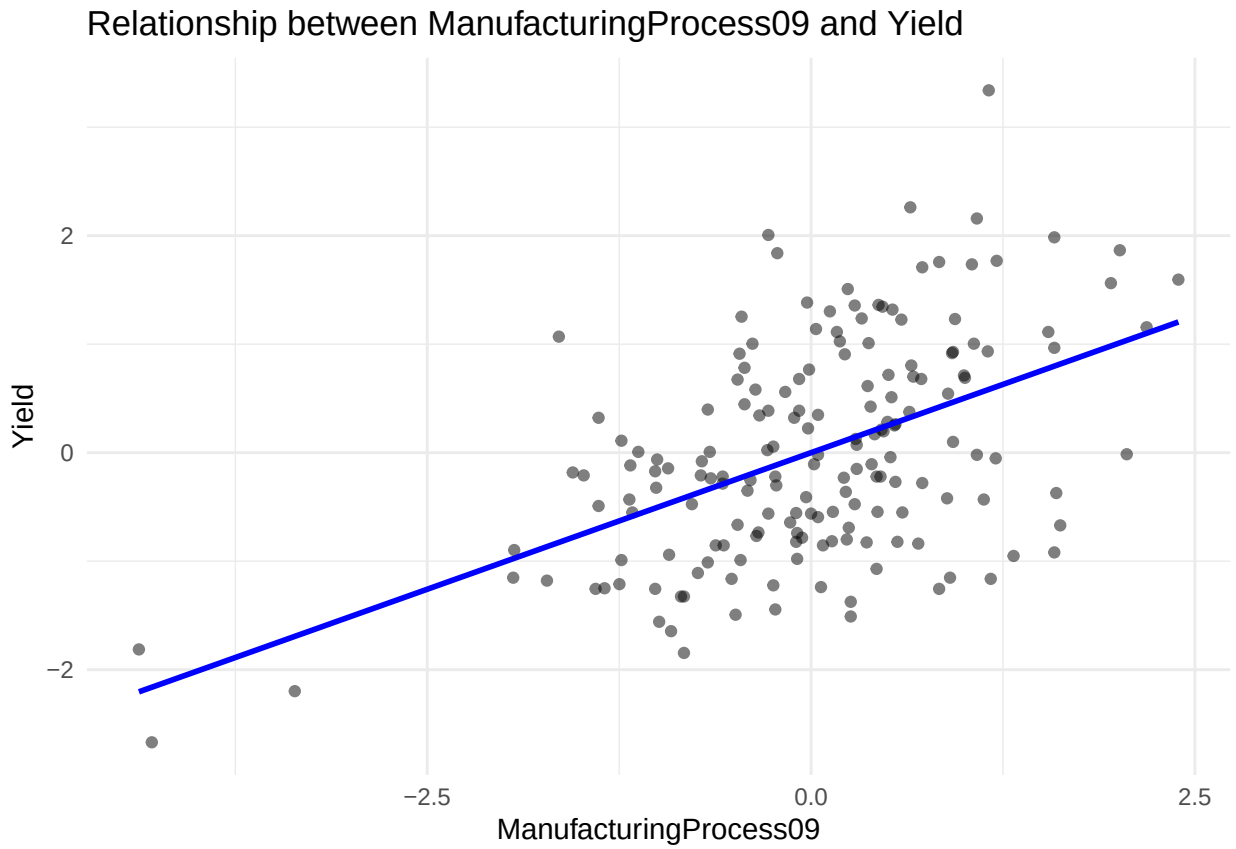
```
# Part (c): Plotting Relationships Between Top Predictors and Yield
# List of most important predictors for analysis
top_features <- c("ManufacturingProcess32", "ManufacturingProcess09", "ManufacturingProcess13",
                  "ManufacturingProcess39", "ManufacturingProcess22", "ManufacturingProcess28",
                  "BiologicalMaterial12", "BiologicalMaterial03", "ManufacturingProcess01",
                  "ManufacturingProcess33")

# Generate scatter plots with linear fit for each selected predictor
for (feature in top_features) {
  plot_data <- data.frame(X = predictors[[feature]], Y = response)

  plot <- ggplot(plot_data, aes_string(x = "X", y = "Y")) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "lm", color = "blue", se = FALSE) +
    labs(
      title = paste("Relationship between", feature, "and Yield"),
      x = feature, y = "Yield"
    ) +
    theme_minimal()

  print(plot)
}
```

```
## 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.
```

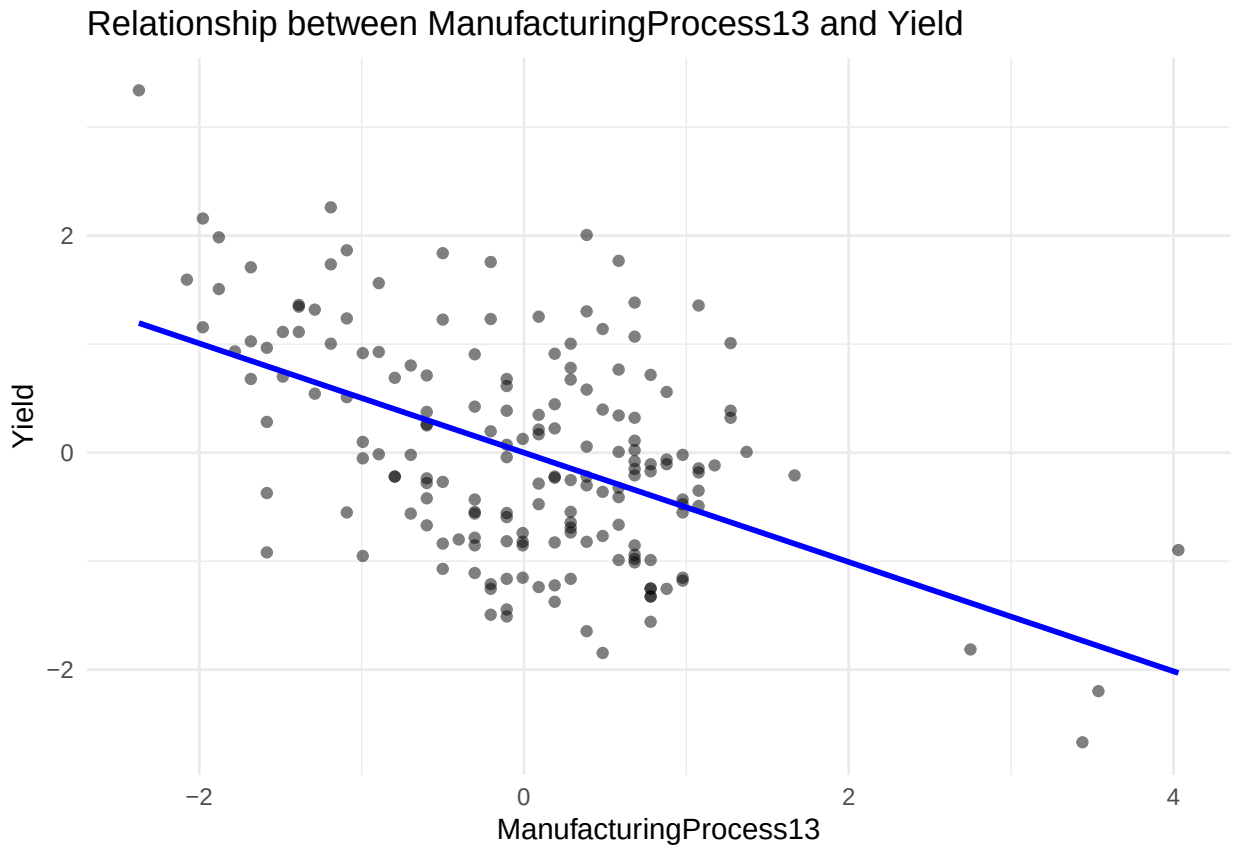
```
## `geom_smooth()` using formula = 'y ~ x'
```



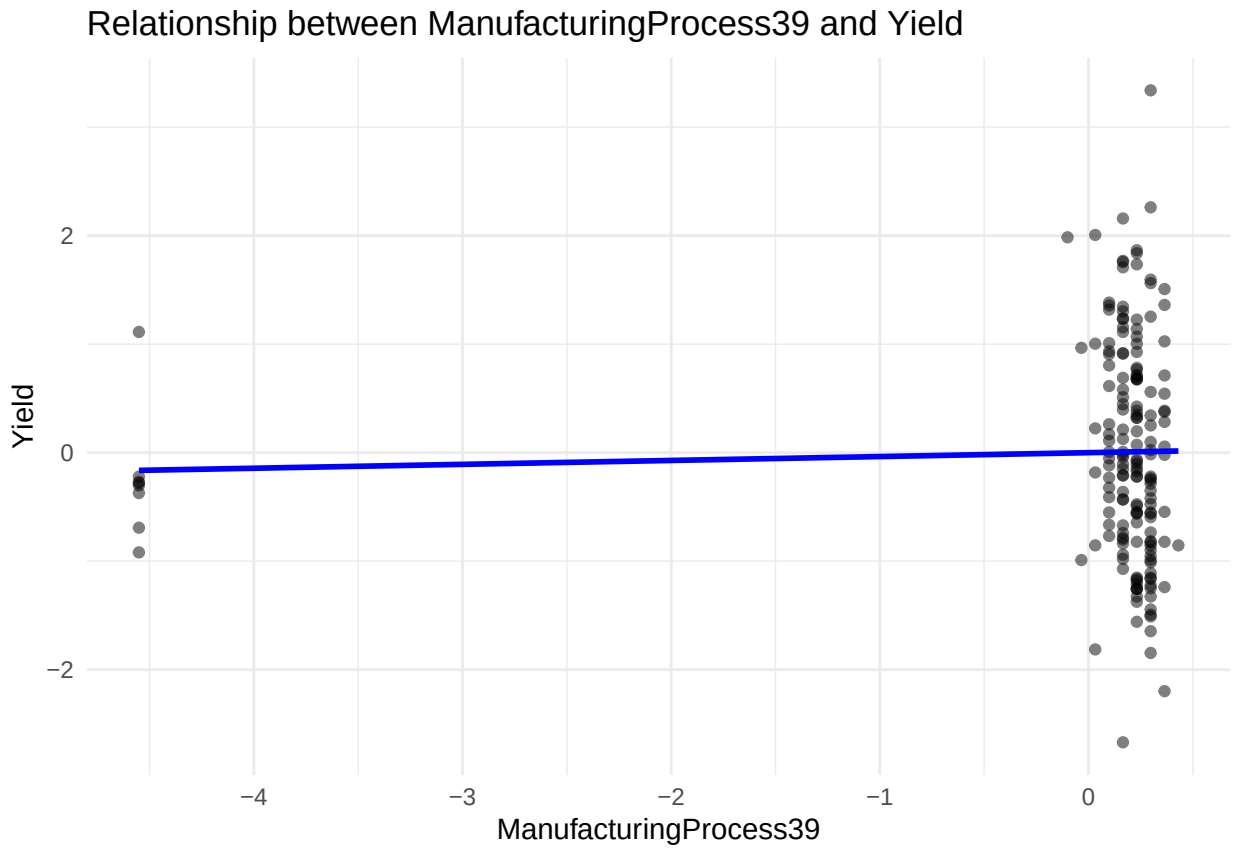

```
## `geom_smooth()` using formula = 'y ~ x'
```



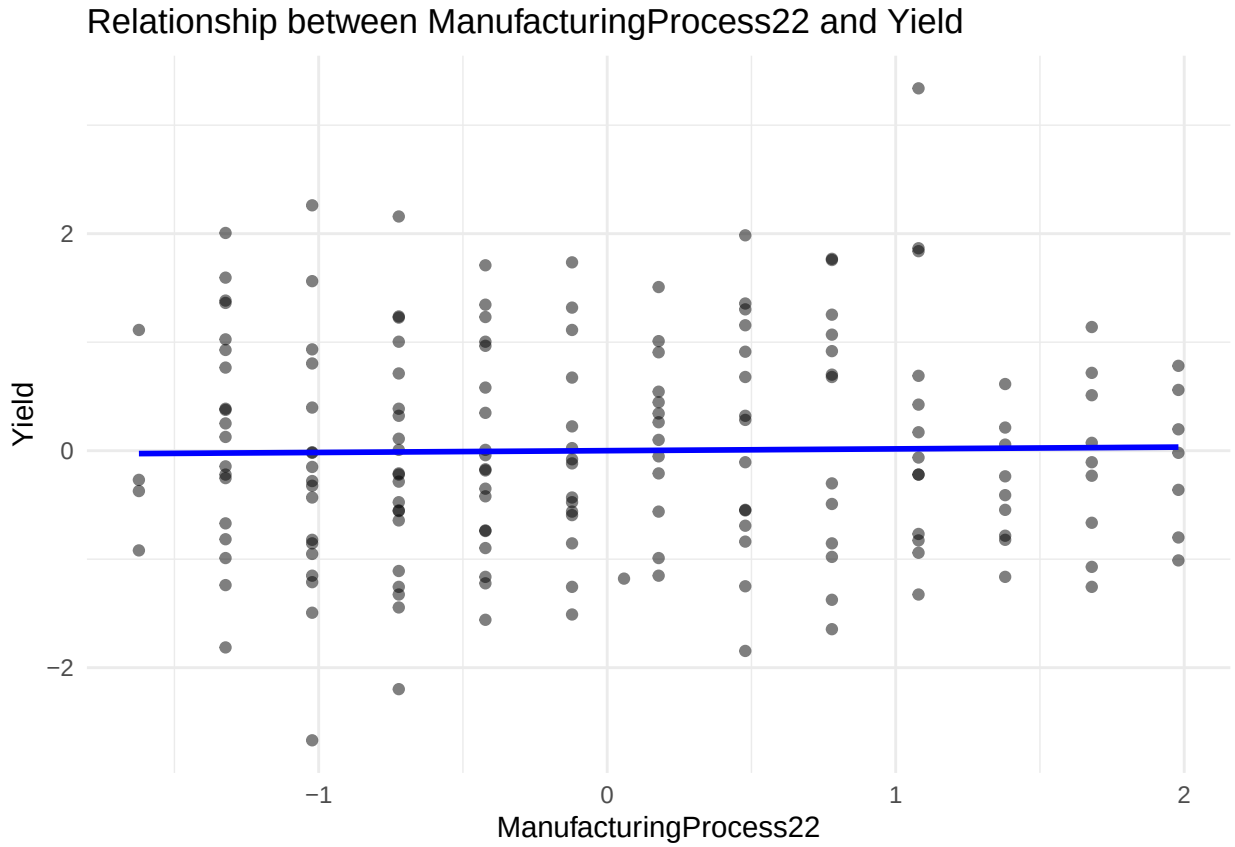

```
## `geom_smooth()` using formula = 'y ~ x'
```



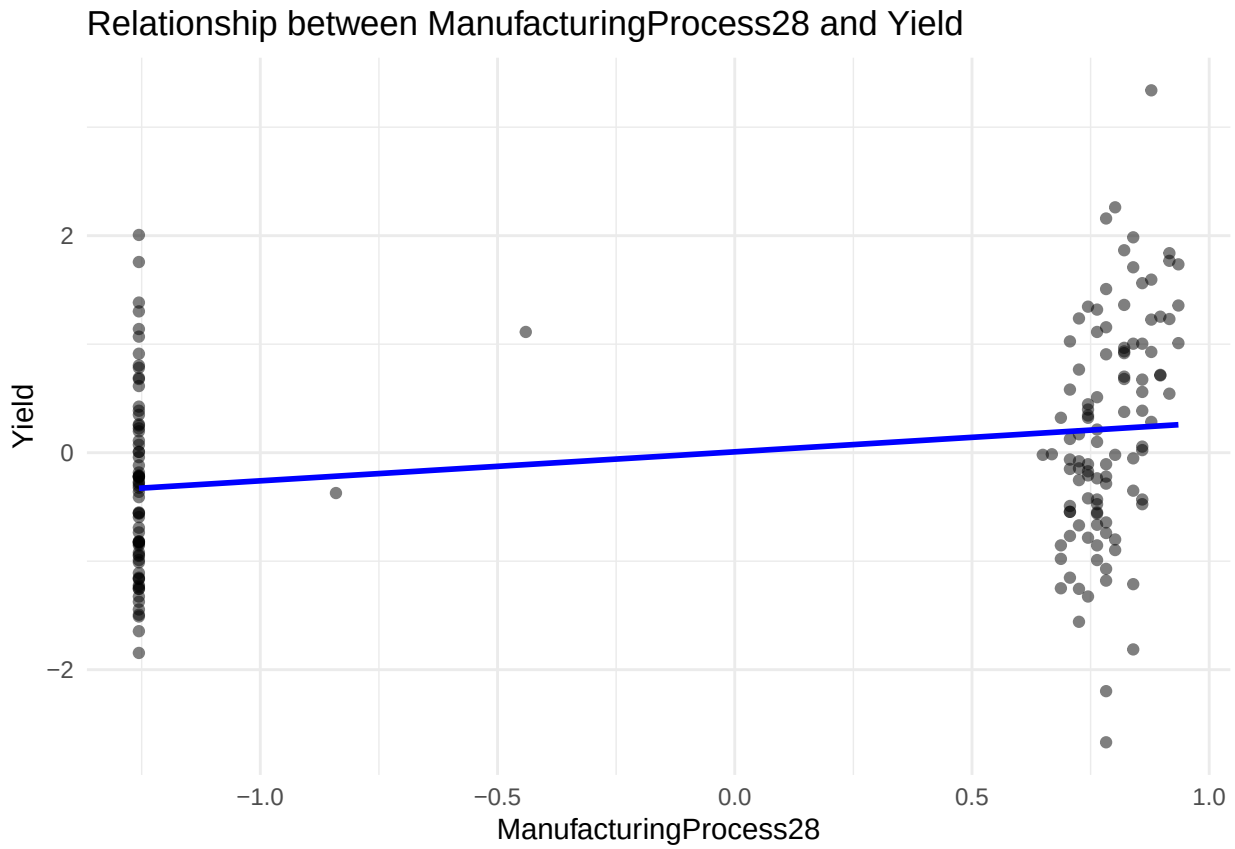

```
## `geom_smooth()` using formula = 'y ~ x'
```



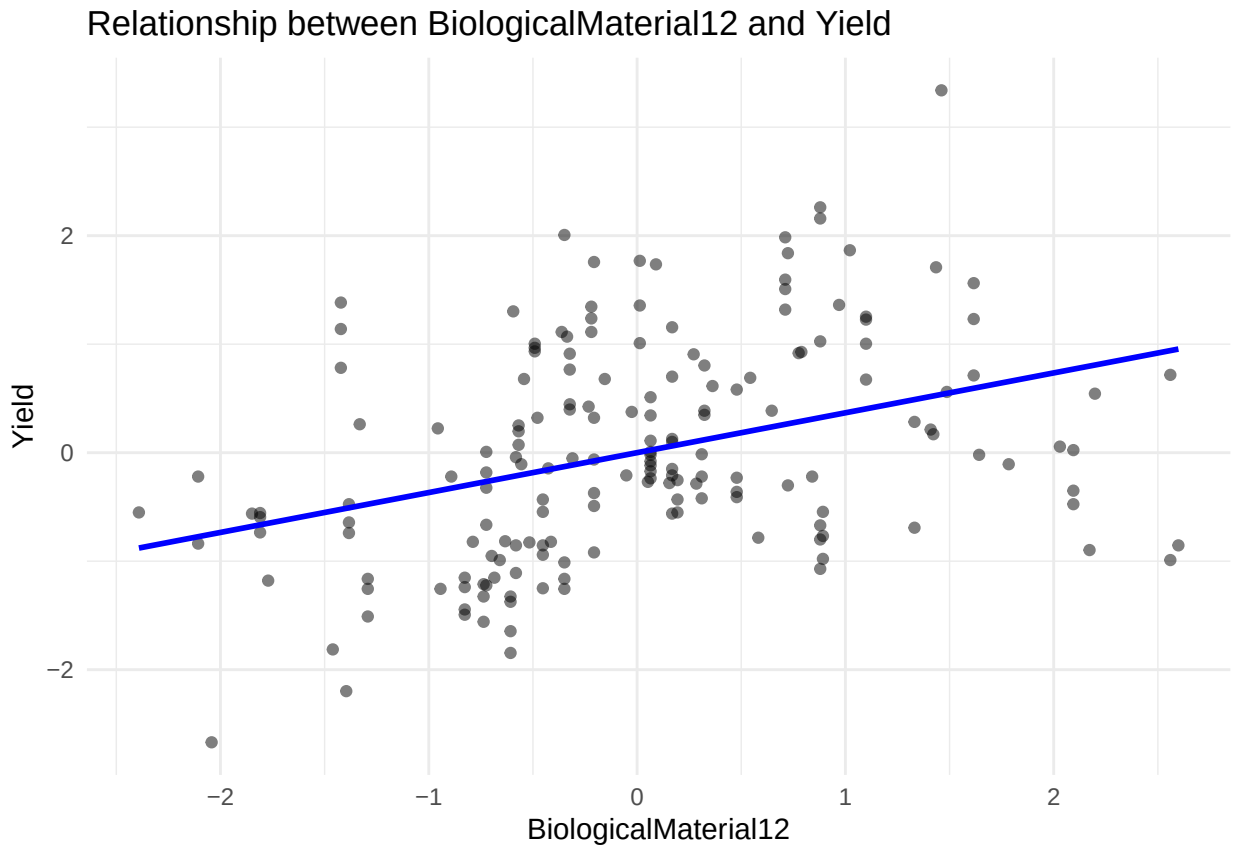

```
## `geom_smooth()` using formula = 'y ~ x'
```



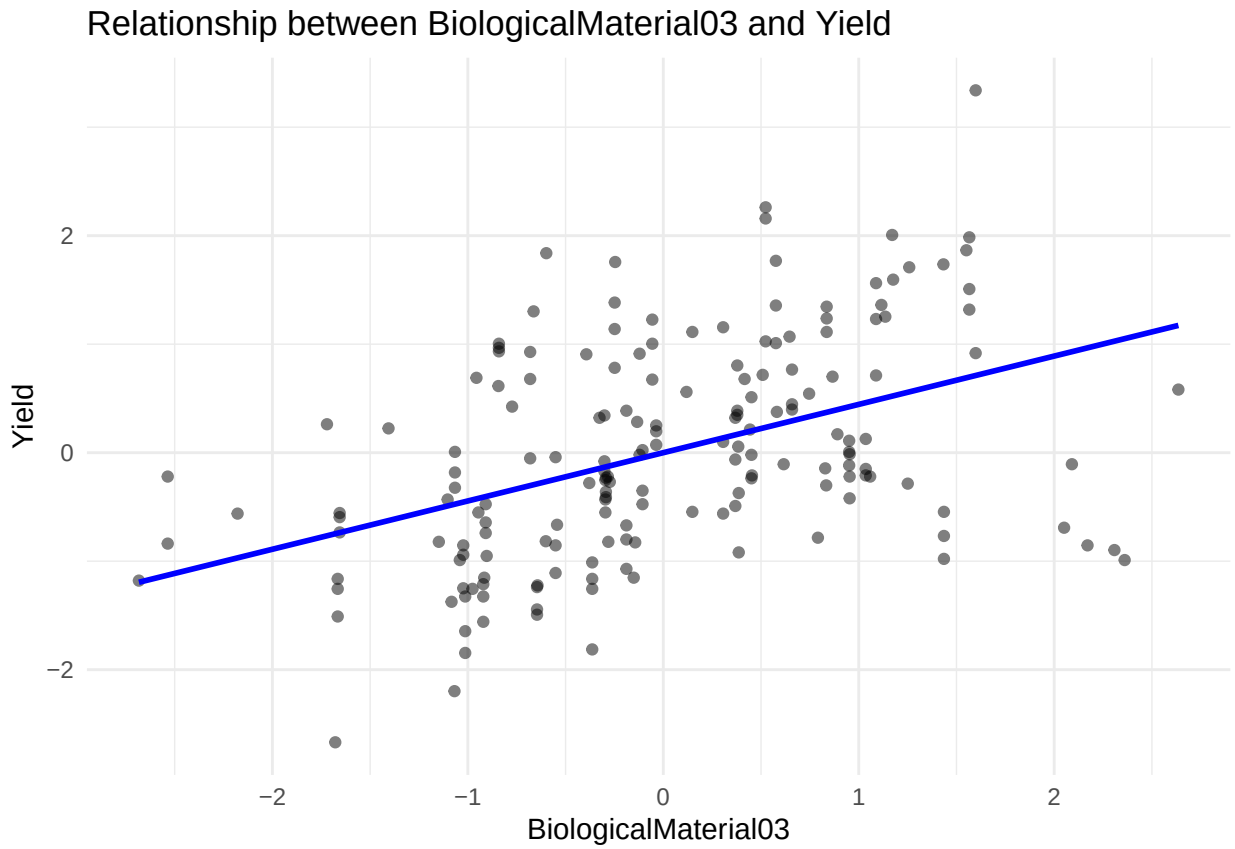

```
## `geom_smooth()` using formula = 'y ~ x'
```



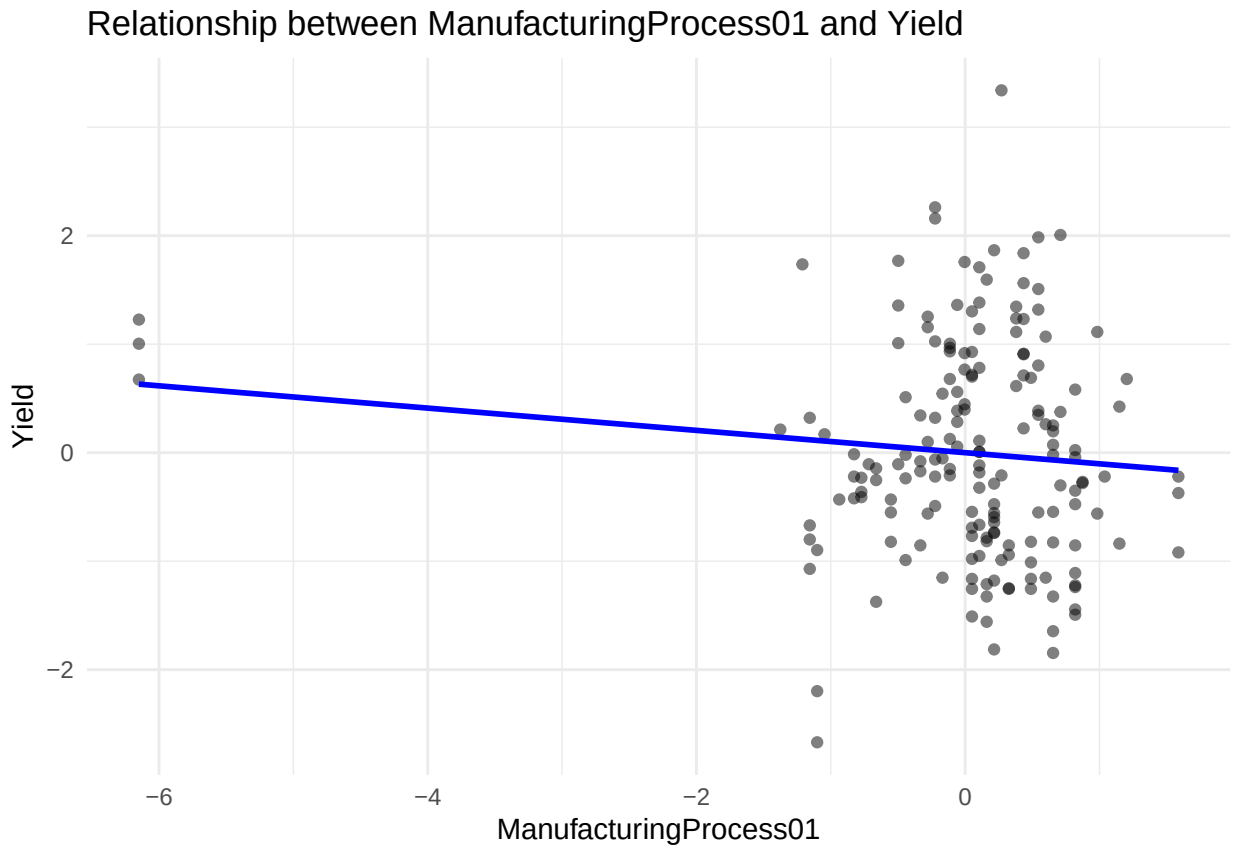

```
## `geom_smooth()` using formula = 'y ~ x'
```



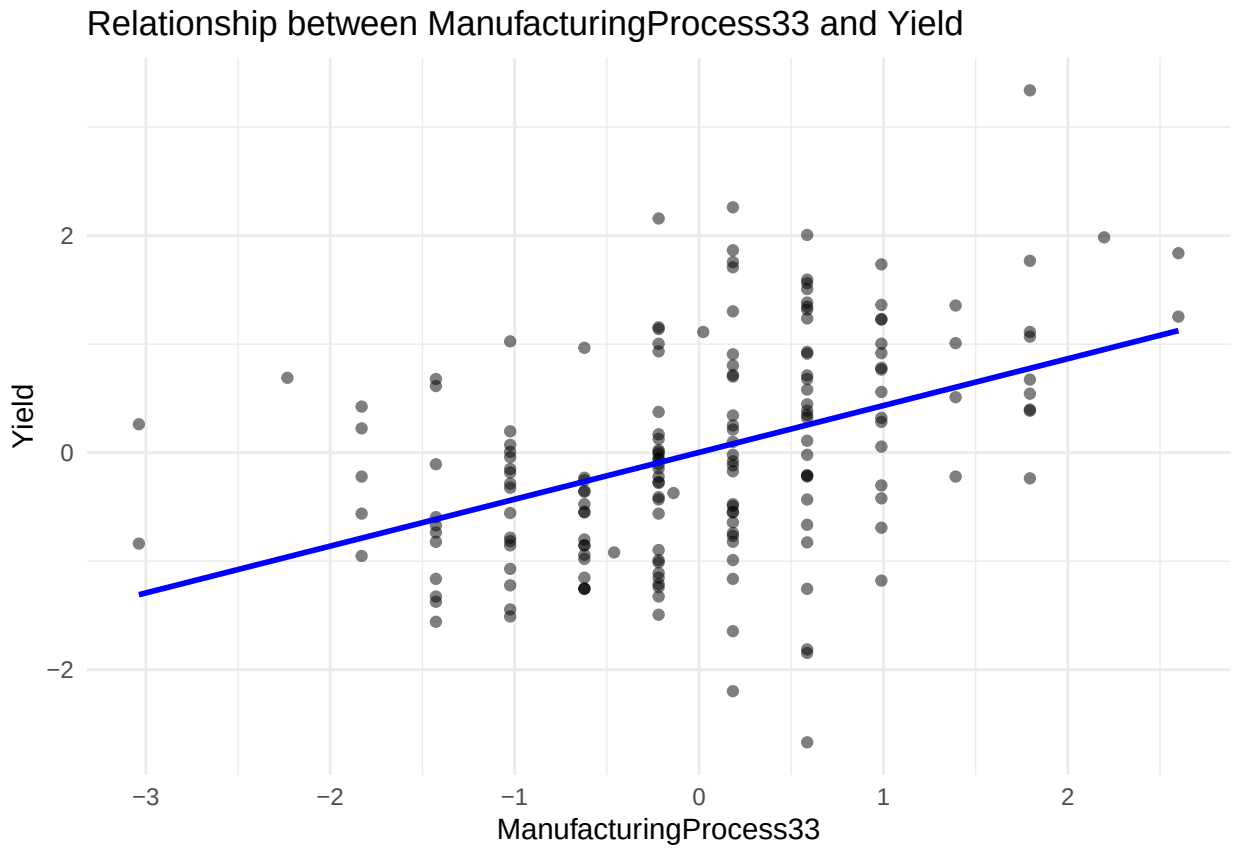

```
## `geom_smooth()` using formula = 'y ~ x'
```




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




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




```
# Output: Displays scatter plots with a linear trend line, illustrating the relationship between each t
```