ShovanBiswas-DATA24_Homework_08

Shovan Biswas

10/30/2020

Libraries

```
library(tidyverse)
library(kableExtra)
library(corrplot)
library(reshape2)
library(Amelia)
library(dlookr)
library(fpp2)
library(plotly)
library(gridExtra)
library(readxl)
library(ggplot2)
library(urca)
library(tseries)
library(AppliedPredictiveModeling)
library(RANN)
library(psych)
library(e1071)
library(corrplot)
library(glmnet)
library(mlbench)
library(caret)
library(earth)
```

Exercise 7.2

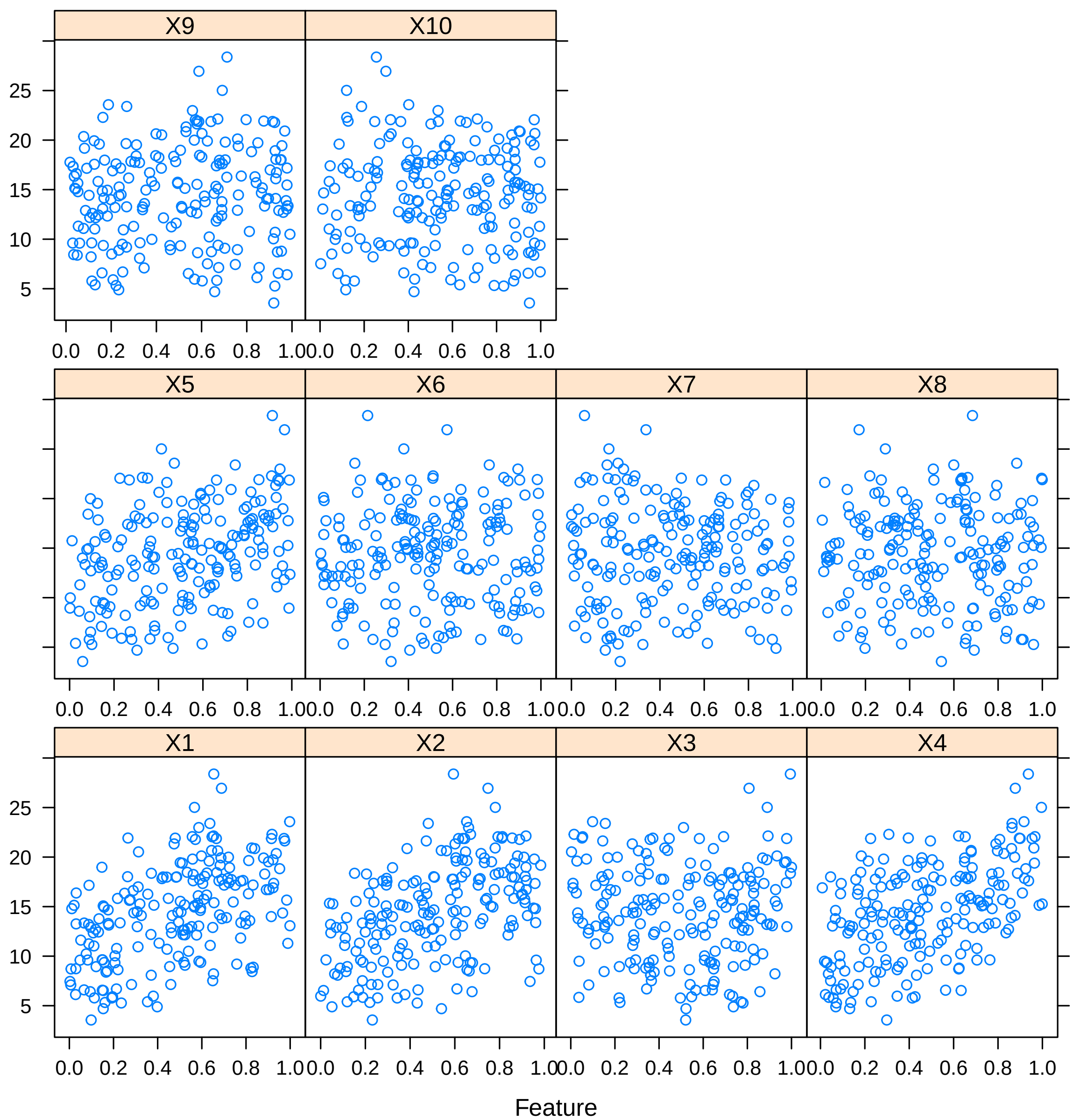
Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data:

$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + N(0, \sigma^2)$

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:

(This exercise is based on library(mlbench), which I included in libraries at the top.)

```r
set.seed(200)

trainingData <- mlbench.friedman1(200, sd = 1)

## We convert the 'x' data from a matrix to a data frame
## 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.
```

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

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

Tune several models on these data. For example:

(I included library caret in libraries at the top.)

```r
knnModel <- train(x = trainingData$x, y = trainingData$y, method = "knn", preProc = c("center", "scale")
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
## perforamnce values
postResample(pred = knnPred, obs = testData$y)
```

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

Which models appear to give the best performance? Does MARS select the informative predictors (those named X1–X5)?

Answer:

We observed above that the RMSE of kNN model is 3.2932153. In the following, we'll explore all other models, Neural Networks, MARS and SVM, mentioned in the book, in this order.

Neural Networks

Used code from page 163 of textbook.

```
nnetGrid <- expand.grid(.decay = c(0, 0.01, .1), .size = c(1:10), .bag = FALSE)

nnetTune <- train(x = trainingData$x, y = trainingData$y, method = "avNNet", tuneGrid = nnetGrid,
                  trControl = trainControl(method = "cv"), preProc = c("center", "scale"),
                  linout = TRUE, trace = FALSE,
                  MaxNWts = 10 * (ncol(trainingData$x) + 1) + 10 + 1,
                  maxit = 500)
```

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

```
nnetTune
```

```
## Model Averaged Neural Network
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE      Rsquared   MAE
##   0.00    1    2.434845  0.7683498  1.921367
##   0.00    2    2.497822  0.7558233  1.993325
##   0.00    3    2.037885  0.8419795  1.609413
##   0.00    4    1.900063  0.8584928  1.529545
##   0.00    5    2.176661  0.8092998  1.628603
##   0.00    6    2.743381  0.7255103  1.988222
##   0.00    7    3.496229  0.6401273  2.493454
##   0.00    8    4.034891  0.5941657  2.749735
##   0.00    9    4.221796  0.5137164  2.800450
##   0.00   10    4.682342  0.5848908  2.818883
##   0.01    1    2.437231  0.7689665  1.934978
##   0.01    2    2.510986  0.7596191  1.988260
##   0.01    3    1.999944  0.8419567  1.555751
##   0.01    4    2.003357  0.8445288  1.549723
##   0.01    5    2.104801  0.8296459  1.664982
##   0.01    6    2.314704  0.7997307  1.857949
##   0.01    7    2.341101  0.8072335  1.872758
##   0.01    8    2.205611  0.8163107  1.748153
##   0.01    9    2.262921  0.8146166  1.776693
##   0.01   10    2.453311  0.7709666  1.981977
```

```
##   0.10    1    2.450897  0.7652309  1.942945
##   0.10    2    2.489399  0.7606443  1.997060
##   0.10    3    2.200693  0.8155496  1.786599
##   0.10    4    2.059322  0.8432340  1.651716
##   0.10    5    2.189025  0.8133603  1.729453
##   0.10    6    2.215091  0.8128993  1.757966
##   0.10    7    2.209521  0.8196474  1.786772
##   0.10    8    2.317124  0.8010433  1.826655
##   0.10    9    2.286711  0.7928430  1.849002
##   0.10   10    2.238560  0.8113030  1.787851
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 4, decay = 0 and bag = FALSE.
```

```r
Neural_pred <- postResample(pred = predict(nnetTune, newdata = testData$x), obs = testData$y)
Neural_pred
```

```
##     RMSE Rsquared      MAE
## 2.496722 0.784618 1.685182
```

Observation: RMSE of Neural Networks is 2.496722. It's way higher than what we obtained in kNN (3.2040595).

```r
varImp(nnetTune)
```

```
## loess r-squared variable importance
##
##      Overall
## X4  100.0000
## X1   95.5047
## X2   89.6186
## X5   45.2170
## X3   29.9330
## X9    6.3299
## X10   5.5182
## X8    3.2527
## X6    0.8884
## X7    0.0000
```

The top 5 variables are X4, X1, X2, X5, X3.

MARS

Used code from page 165 of textbook. Included library(earth) in libraries at the top.

```r
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)

marsTuned <- train(x = trainingData$x, y = trainingData$y, method = "earth", tuneGrid = marsGrid,
                   trControl = trainControl(method = "cv"))

marsTuned
```

```
## 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      4.334325  0.2599883  3.607719
##   1        3      3.599334  0.4805557  2.888987
##   1        4      2.637145  0.7290848  2.087677
##   1        5      2.283872  0.7939684  1.817343
##   1        6      2.125875  0.8183677  1.647491
##   1        7      1.766013  0.8733619  1.410328
##   1        8      1.671282  0.8842102  1.324258
##   1        9      1.645406  0.8867947  1.322041
##   1       10      1.597968  0.8926582  1.297518
##   1       11      1.540109  0.8996361  1.237949
##   1       12      1.545349  0.8992979  1.243771
##   1       13      1.535169  0.9010122  1.233571
##   1       14      1.529405  0.9018457  1.223874
##   1       15      1.529405  0.9018457  1.223874
##   1       16      1.529405  0.9018457  1.223874
##   1       17      1.529405  0.9018457  1.223874
##   1       18      1.529405  0.9018457  1.223874
##   1       19      1.529405  0.9018457  1.223874
##   1       20      1.529405  0.9018457  1.223874
##   1       21      1.529405  0.9018457  1.223874
##   1       22      1.529405  0.9018457  1.223874
##   1       23      1.529405  0.9018457  1.223874
##   1       24      1.529405  0.9018457  1.223874
##   1       25      1.529405  0.9018457  1.223874
##   1       26      1.529405  0.9018457  1.223874
##   1       27      1.529405  0.9018457  1.223874
##   1       28      1.529405  0.9018457  1.223874
##   1       29      1.529405  0.9018457  1.223874
##   1       30      1.529405  0.9018457  1.223874
##   1       31      1.529405  0.9018457  1.223874
##   1       32      1.529405  0.9018457  1.223874
##   1       33      1.529405  0.9018457  1.223874
##   1       34      1.529405  0.9018457  1.223874
##   1       35      1.529405  0.9018457  1.223874
##   1       36      1.529405  0.9018457  1.223874
##   1       37      1.529405  0.9018457  1.223874
##   1       38      1.529405  0.9018457  1.223874
##   2        2      4.334325  0.2599883  3.607719
##   2        3      3.599334  0.4805557  2.888987
##   2        4      2.637145  0.7290848  2.087677
##   2        5      2.271844  0.7927888  1.823675
##   2        6      2.114868  0.8200184  1.659485
##   2        7      1.780140  0.8733216  1.429346
```

```
##   2        8      1.663164  0.8891928  1.294968
##   2        9      1.460976  0.9122520  1.180387
##   2       10      1.399692  0.9175376  1.122526
##   2       11      1.380002  0.9216251  1.110556
##   2       12      1.312883  0.9284253  1.063321
##   2       13      1.285612  0.9343029  1.014216
##   2       14      1.328520  0.9286650  1.052185
##   2       15      1.322954  0.9298515  1.045527
##   2       16      1.341454  0.9283961  1.053190
##   2       17      1.344590  0.9280972  1.054209
##   2       18      1.340821  0.9285264  1.050274
##   2       19      1.340821  0.9285264  1.050274
##   2       20      1.340821  0.9285264  1.050274
##   2       21      1.340821  0.9285264  1.050274
##   2       22      1.340821  0.9285264  1.050274
##   2       23      1.340821  0.9285264  1.050274
##   2       24      1.340821  0.9285264  1.050274
##   2       25      1.340821  0.9285264  1.050274
##   2       26      1.340821  0.9285264  1.050274
##   2       27      1.340821  0.9285264  1.050274
##   2       28      1.340821  0.9285264  1.050274
##   2       29      1.340821  0.9285264  1.050274
##   2       30      1.340821  0.9285264  1.050274
##   2       31      1.340821  0.9285264  1.050274
##   2       32      1.340821  0.9285264  1.050274
##   2       33      1.340821  0.9285264  1.050274
##   2       34      1.340821  0.9285264  1.050274
##   2       35      1.340821  0.9285264  1.050274
##   2       36      1.340821  0.9285264  1.050274
##   2       37      1.340821  0.9285264  1.050274
##   2       38      1.340821  0.9285264  1.050274
##
## 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
MARS_pred <- postResample(pred = predict(marsTuned, newdata = testData$x), obs = testData$y)
MARS_pred
```

```
##      RMSE  Rsquared       MAE
## 1.2803060 0.9335241 1.0168673
```

Observation: RMSE of MARS is 1.2803060. It's least and best so far.

```r
varImp(marsTuned)
```

```
## earth variable importance
##
##    Overall
## X1  100.00
## X4   75.33
## X2   48.88
## X5   15.63
## X3    0.00
```

The top 5 variables are X1, X4, X2, X5, X3.

Support Vector Machines

Used code from page 167 of textbook.

```
svmRTuned <- train(x = trainingData$x, y = trainingData$y,
                   method = "svmRadial", preProc = c("center", "scale"),
                   tuneLength = 14, trControl = trainControl(method = "cv"))

svmRTuned
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   C        RMSE      Rsquared   MAE
##      0.25  2.504105  0.7940789  1.987142
##      0.50  2.219946  0.8148914  1.750249
##      1.00  2.028115  0.8388693  1.590383
##      2.00  1.899331  0.8561464  1.486326
##      4.00  1.815632  0.8669708  1.424246
##      8.00  1.798299  0.8702910  1.427678
##     16.00  1.797165  0.8702715  1.431259
##     32.00  1.795246  0.8705225  1.429235
##     64.00  1.795246  0.8705225  1.429235
##    128.00  1.795246  0.8705225  1.429235
##    256.00  1.795246  0.8705225  1.429235
##    512.00  1.795246  0.8705225  1.429235
##   1024.00  1.795246  0.8705225  1.429235
##   2048.00  1.795246  0.8705225  1.429235
##
## Tuning parameter 'sigma' was held constant at a value of 0.06104815
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.06104815 and C = 32.
```

```
SVM_pred <- postResample(pred = predict(svmRTuned, newdata = testData$x), obs = testData$y)
SVM_pred
```

```
##      RMSE  Rsquared       MAE
## 2.0693488 0.8263553 1.5718972
```

Observation: RMSE of SVM is 2.0469184.

```r
varImp(svmRTuned)
```

```
## loess r-squared variable importance
##
##      Overall
## X4  100.0000
## X1   95.5047
## X2   89.6186
## X5   45.2170
## X3   29.9330
## X9    6.3299
## X10   5.5182
## X8    3.2527
## X6    0.8884
## X7    0.0000
```

The top 5 variables are X4, X1, X2, X5, X3.

Summary

```r
results <- data.frame(t(postResample(pred = knnPred, obs = testData$y))) %>% mutate("Model" = "KNN")

results <- data.frame(t(Neural_pred)) %>% mutate("Model"= "Neural Networks") %>% bind_rows(results)

results <- data.frame(t(MARS_pred)) %>% mutate("Model"= "MARS") %>% bind_rows(results)

results <- data.frame(t(SVM_pred)) %>% mutate("Model"= "Support Vector Machines") %>% bind_rows(results)
```

```r
results %>% select(Model, RMSE, Rsquared, MAE) %>% arrange(RMSE)
```

```
##                     Model     RMSE  Rsquared      MAE
## 1                    MARS 1.280306 0.9335241 1.016867
## 2 Support Vector Machines 2.069349 0.8263553 1.571897
## 3         Neural Networks 2.496722 0.7846180 1.685182
## 4                     KNN 3.204059 0.6819919 2.568346
```

Conclusion

MARS outperformed the others.

Exercise 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.

Answer:

Before we begin answring the questions, let's pre-process the data. In previous Homework 7, this data had to be imputed and Box-coxed. I'll do the same this time, but will not inspect with histograms, because we already know about that.

```
data(ChemicalManufacturingProcess)
dim(ChemicalManufacturingProcess)
```

```
## [1] 176  58
```

I'll impute the data using the same technique i.e. mice().

```
data_imputed <- mice(ChemicalManufacturingProcess, m = 1, method = "pmm", print = F) %>% complete()
```

```
## Warning: Number of logged events: 135
```

```
any(is.na(data_imputed))
```

```
## [1] FALSE
```

At this point, the data is imputed. I'll proceed to Box-Cox it.

```
# Initially identifying columns, whose skewness are not less than 1.
transform_cols <- c()

for(i in seq(from = 1, to = length(data_imputed), by = 1)) {
  if(abs(skewness(data_imputed[, i])) >= 1) {
    transform_cols <- append(transform_cols, names(data_imputed[i]))
  }
}

# Applying Box-cox.
lambda <- NULL
data_imputed_2 <- data_imputed

for (i in 1:length(transform_cols)) {
  lambda[transform_cols[i]] <- BoxCox.lambda(abs(data_imputed[, transform_cols[i]]))

  data_imputed_2[c(transform_cols[i])] <- BoxCox(data_imputed[transform_cols[i]], lambda[transform_cols
}
```

At this point, the data is pre-processed. The pre-processed data is stored in the variable data_imputed_2.

So, I'll proceed to split the data into train and test in 80/20 ratio.

```
set.seed(200)

split_index <- createDataPartition(data_imputed_2$Yield, p = 0.8, list = FALSE)
X_train <- data_imputed_2[split_index, ]
y_train <- data_imputed_2$Yield[split_index]

X_test <- data_imputed_2[-split_index, ]
y_test <- data_imputed_2$Yield[-split_index]
```

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

kNN

```
knnModel <- train(x = X_train, y = y_train, method = "knn", preProc = c("center", "scale"), tuneLength =
knnModel
```

```
## k-Nearest Neighbors
##
## 144 samples
##  58 predictor
##
## Pre-processing: centered (58), scaled (58)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 144, 144, 144, 144, 144, 144, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  1.199978  0.5674420  0.9410336
##    7  1.194618  0.5731676  0.9391985
##    9  1.205958  0.5669943  0.9449103
##   11  1.219228  0.5582151  0.9566777
##   13  1.239287  0.5477906  0.9751440
##   15  1.259243  0.5350655  0.9937786
##   17  1.263582  0.5390212  1.0026756
##   19  1.276911  0.5347938  1.0111498
##   21  1.280863  0.5389095  1.0153793
##   23  1.295654  0.5329888  1.0280432
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

```
knnPred <- postResample(pred = predict(knnModel, newdata = X_test), obs = y_test)
knnPred
```

```
##      RMSE  Rsquared       MAE
## 1.3655892 0.6965463 1.1388839
```

Neural Networks

```
set.seed(200)

nnetGrid <- expand.grid(.decay = c(0, 0.01, .1), .size = c(1:10), .bag = FALSE)

nnetTune <- train(x = X_train,
                  y = y_train,
                  method = "avNNet",
                  tuneGrid = nnetGrid,
                  trControl = trainControl(method = "cv"),
```

```r
                   preProc = c("center", "scale"),
                   linout = TRUE,
                   trace = FALSE,
                   MaxNWts = 10 * (ncol(X_train) + 1) + 10 + 1,
                   maxit = 500)

nnetTune
```

```
## Model Averaged Neural Network
##
## 144 samples
##  58 predictor
##
## Pre-processing: centered (58), scaled (58)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 129, 129, 130, 131, 131, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE       Rsquared   MAE
##   0.00    1    1.4206086  0.4771598  1.1636579
##   0.00    2    1.3464206  0.4910898  1.0903323
##   0.00    3    1.0512727  0.7334446  0.8398412
##   0.00    4    1.2540153  0.6370139  1.0041790
##   0.00    5    1.6600161  0.5195195  1.3510075
##   0.00    6    1.9905281  0.4925013  1.5050869
##   0.00    7    2.7225197  0.4100306  2.0331524
##   0.00    8    3.2422674  0.4568998  2.4683062
##   0.00    9    4.8528902  0.2649851  3.5101438
##   0.00   10    5.0031116  0.2364041  3.5836195
##   0.01    1    0.3779699  0.9323267  0.1715984
##   0.01    2    0.5999804  0.8926036  0.3427568
##   0.01    3    0.7226971  0.8747782  0.4506862
##   0.01    4    1.1599770  0.7294355  0.7242155
##   0.01    5    1.0446853  0.7880058  0.7435489
##   0.01    6    1.1963253  0.7357865  0.8240162
##   0.01    7    1.2206717  0.6564380  0.8932235
##   0.01    8    1.1463384  0.7296851  0.8880410
##   0.01    9    1.7504617  0.5922516  1.3301758
##   0.01   10    2.2169413  0.4798770  1.6104557
##   0.10    1    0.4217549  0.9441752  0.2964366
##   0.10    2    1.0663106  0.7886952  0.6532342
##   0.10    3    1.2924731  0.7460900  0.8011197
##   0.10    4    1.1894907  0.7372154  0.7096645
##   0.10    5    1.0203844  0.7931394  0.6330131
##   0.10    6    1.4086756  0.6958338  0.7999990
##   0.10    7    1.4307703  0.6782201  0.8510574
##   0.10    8    1.3001500  0.6632119  0.8822045
##   0.10    9    1.2489547  0.6619679  0.9037666
##   0.10   10    1.0780311  0.7499390  0.8094006
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 1, decay = 0.01 and bag = FALSE.
```

```r
Neural_pred <- postResample(pred = predict(nnetTune, newdata = X_test), obs = y_test)
Neural_pred
```

```
##      RMSE  Rsquared       MAE
## 0.3997863 0.9664705 0.1782120
```

MARS

```r
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)

marsTuned <- train(x = X_train, y = y_train, method = "earth", tuneGrid = marsGrid, preProc = c("center

marsTuned
```

```
## Multivariate Adaptive Regression Spline
##
## 144 samples
##  58 predictor
##
## Pre-processing: centered (58), scaled (58)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 131, 130, 130, 128, 130, 130, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE          Rsquared  MAE
##   1        2      9.457448e-15  1         9.179015e-15
##   1        3      9.457448e-15  1         9.179015e-15
##   1        4      9.457448e-15  1         9.179015e-15
##   1        5      9.457448e-15  1         9.179015e-15
##   1        6      9.457448e-15  1         9.179015e-15
##   1        7      9.457448e-15  1         9.179015e-15
##   1        8      9.457448e-15  1         9.179015e-15
##   1        9      9.457448e-15  1         9.179015e-15
##   1       10      9.457448e-15  1         9.179015e-15
##   1       11      9.457448e-15  1         9.179015e-15
##   1       12      9.457448e-15  1         9.179015e-15
##   1       13      9.457448e-15  1         9.179015e-15
##   1       14      9.457448e-15  1         9.179015e-15
##   1       15      9.457448e-15  1         9.179015e-15
##   1       16      9.457448e-15  1         9.179015e-15
##   1       17      9.457448e-15  1         9.179015e-15
##   1       18      9.457448e-15  1         9.179015e-15
##   1       19      9.457448e-15  1         9.179015e-15
##   1       20      9.457448e-15  1         9.179015e-15
##   1       21      9.457448e-15  1         9.179015e-15
##   1       22      9.457448e-15  1         9.179015e-15
##   1       23      9.457448e-15  1         9.179015e-15
##   1       24      9.457448e-15  1         9.179015e-15
##   1       25      9.457448e-15  1         9.179015e-15
##   1       26      9.457448e-15  1         9.179015e-15
##   1       27      9.457448e-15  1         9.179015e-15
```

```
##   1       28      9.457448e-15  1         9.179015e-15
##   1       29      9.457448e-15  1         9.179015e-15
##   1       30      9.457448e-15  1         9.179015e-15
##   1       31      9.457448e-15  1         9.179015e-15
##   1       32      9.457448e-15  1         9.179015e-15
##   1       33      9.457448e-15  1         9.179015e-15
##   1       34      9.457448e-15  1         9.179015e-15
##   1       35      9.457448e-15  1         9.179015e-15
##   1       36      9.457448e-15  1         9.179015e-15
##   1       37      9.457448e-15  1         9.179015e-15
##   1       38      9.457448e-15  1         9.179015e-15
##   2        2      9.457448e-15  1         9.179015e-15
##   2        3      9.457448e-15  1         9.179015e-15
##   2        4      9.457448e-15  1         9.179015e-15
##   2        5      9.457448e-15  1         9.179015e-15
##   2        6      9.457448e-15  1         9.179015e-15
##   2        7      9.457448e-15  1         9.179015e-15
##   2        8      9.457448e-15  1         9.179015e-15
##   2        9      9.457448e-15  1         9.179015e-15
##   2       10      9.457448e-15  1         9.179015e-15
##   2       11      9.457448e-15  1         9.179015e-15
##   2       12      9.457448e-15  1         9.179015e-15
##   2       13      9.457448e-15  1         9.179015e-15
##   2       14      9.457448e-15  1         9.179015e-15
##   2       15      9.457448e-15  1         9.179015e-15
##   2       16      9.457448e-15  1         9.179015e-15
##   2       17      9.457448e-15  1         9.179015e-15
##   2       18      9.457448e-15  1         9.179015e-15
##   2       19      9.457448e-15  1         9.179015e-15
##   2       20      9.457448e-15  1         9.179015e-15
##   2       21      9.457448e-15  1         9.179015e-15
##   2       22      9.457448e-15  1         9.179015e-15
##   2       23      9.457448e-15  1         9.179015e-15
##   2       24      9.457448e-15  1         9.179015e-15
##   2       25      9.457448e-15  1         9.179015e-15
##   2       26      9.457448e-15  1         9.179015e-15
##   2       27      9.457448e-15  1         9.179015e-15
##   2       28      9.457448e-15  1         9.179015e-15
##   2       29      9.457448e-15  1         9.179015e-15
##   2       30      9.457448e-15  1         9.179015e-15
##   2       31      9.457448e-15  1         9.179015e-15
##   2       32      9.457448e-15  1         9.179015e-15
##   2       33      9.457448e-15  1         9.179015e-15
##   2       34      9.457448e-15  1         9.179015e-15
##   2       35      9.457448e-15  1         9.179015e-15
##   2       36      9.457448e-15  1         9.179015e-15
##   2       37      9.457448e-15  1         9.179015e-15
##   2       38      9.457448e-15  1         9.179015e-15
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 2 and degree = 1.
```

```r
MARS_pred <- postResample(pred = predict(marsTuned, newdata = X_test), obs = y_test)
MARS_pred
```

```
##         RMSE     Rsquared          MAE
## 7.105427e-15 1.000000e+00 7.105427e-15
```

SVM

```r
svmRTuned <- train(x = X_train, y = y_train,
                   method = "svmRadial", preProc = c("center", "scale"),
                   tuneLength = 14, trControl = trainControl(method = "cv"))

svmRTuned
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 144 samples
##  58 predictor
##
## Pre-processing: centered (58), scaled (58)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 131, 128, 128, 131, 129, 129, ...
## Resampling results across tuning parameters:
##
##   C        RMSE       Rsquared   MAE
##      0.25  1.1039900  0.7361614  0.9016199
##      0.50  0.8706268  0.8337615  0.6961979
##      1.00  0.6976655  0.8951262  0.5425394
##      2.00  0.5816475  0.9226768  0.4553709
##      4.00  0.5682445  0.9247077  0.4462486
##      8.00  0.5682445  0.9247077  0.4462486
##     16.00  0.5682445  0.9247077  0.4462486
##     32.00  0.5682445  0.9247077  0.4462486
##     64.00  0.5682445  0.9247077  0.4462486
##    128.00  0.5682445  0.9247077  0.4462486
##    256.00  0.5682445  0.9247077  0.4462486
##    512.00  0.5682445  0.9247077  0.4462486
##   1024.00  0.5682445  0.9247077  0.4462486
##   2048.00  0.5682445  0.9247077  0.4462486
##
## Tuning parameter 'sigma' was held constant at a value of 0.01219688
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.01219688 and C = 4.
```

```r
SVM_pred <- postResample(pred = predict(svmRTuned, newdata = X_test), obs = y_test)
SVM_pred
```

```
##      RMSE  Rsquared       MAE
## 0.7887036 0.9065050 0.5148177
```

Summary

```r
results <- data.frame(t(knnPred)) %>% mutate("Model" = "KNN")

results <- data.frame(t(Neural_pred)) %>% mutate("Model"= "Neural Networks") %>% bind_rows(results)

results <- data.frame(t(MARS_pred)) %>% mutate("Model"= "MARS") %>% bind_rows(results)

results <- data.frame(t(SVM_pred)) %>% mutate("Model"= "Support Vector Machines") %>% bind_rows(results)
```

```r
results %>% select(Model, RMSE, Rsquared, MAE) %>% arrange(RMSE)
```

```
##                     Model         RMSE  Rsquared          MAE
## 1                    MARS 7.105427e-15 1.0000000 7.105427e-15
## 2         Neural Networks 3.997863e-01 0.9664705 1.782120e-01
## 3 Support Vector Machines 7.887036e-01 0.9065050 5.148177e-01
## 4                     KNN 1.365589e+00 0.6965463 1.138884e+00
```

Conclusion

MARS outperformed the others.

(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?

```r
varImp(marsTuned)
```

```
## earth variable importance
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning -Inf
```

```
##       Overall
## Yield     NaN
```

varImp(marsTuned) did not return any predictor. So, let me try the second best mode i.e. Neural Networks.

```r
varImp(nnetTune)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 58)
##
##                        Overall
## Yield                   100.00
## ManufacturingProcess32   38.65
## BiologicalMaterial06     33.94
## ManufacturingProcess13   30.24
## BiologicalMaterial03     29.55
## BiologicalMaterial12     26.73
## ManufacturingProcess17   26.54
```

```
## ManufacturingProcess09   24.78
## ManufacturingProcess06   22.64
## ManufacturingProcess31   22.03
## ManufacturingProcess36   20.99
## BiologicalMaterial02     20.68
## BiologicalMaterial11     18.81
## ManufacturingProcess33   17.92
## ManufacturingProcess30   16.24
## ManufacturingProcess29   15.64
## ManufacturingProcess11   15.11
## BiologicalMaterial09     14.82
## BiologicalMaterial04     14.82
## BiologicalMaterial08     14.26
```

In the case of Neural Networks, 6 of the top ten predictors are ManufacturingProcess predictors and 3 are BiologicalMaterial. So, the ManufacturingProcess predictors dominate.

PLS_MODEL

```
set.seed(200)
pls_model <- train(x = X_train, y = y_train, method = "pls", preProc = c("center", "scale"), trControl =

pls_prediction <- predict(pls_model, newdata = X_test)

results <- data.frame(Model = "PLS", RMSE = caret::RMSE(pls_prediction, y_test), Rsquared = caret::R2(pl
results
```

```
##   Model       RMSE  Rsquared
## 1   PLS 0.03820394 0.9996693
```

```
varImp(pls_model)
```

```
## Warning: package 'pls' was built under R version 3.6.3

##
## Attaching package: 'pls'

## The following object is masked from 'package:caret':
##
##     R2

## The following object is masked from 'package:fpp2':
##
##     gasoline

## The following object is masked from 'package:corrplot':
##
##     corrplot
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
## pls variable importance
##
##   only 20 most important variables shown (out of 58)
##
##                         Overall
## Yield                    100.00
## ManufacturingProcess32    41.70
## ManufacturingProcess13    36.68
## ManufacturingProcess17    34.87
## ManufacturingProcess09    32.33
## ManufacturingProcess36    29.09
## ManufacturingProcess33    27.87
## BiologicalMaterial02      26.95
## ManufacturingProcess06    26.66
## ManufacturingProcess12    25.99
## BiologicalMaterial06      24.47
## BiologicalMaterial08      24.23
## BiologicalMaterial12      23.19
## BiologicalMaterial04      23.07
## ManufacturingProcess28    22.72
## ManufacturingProcess29    21.34
## BiologicalMaterial11      21.31
## BiologicalMaterial01      21.30
## BiologicalMaterial03      20.58
## ManufacturingProcess04    20.30
```

In nonlinear models, MARS performed best. The RMSE was very close to zero. But PLS_MODEL, which is linear, returned an RMSE of 0.1308365 is higher than MARS's RMSE. So, it faired worse.

However, in linear model PLS_MODEL, among the top 10 variables, ManufacturingProcess is dominant.

(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?

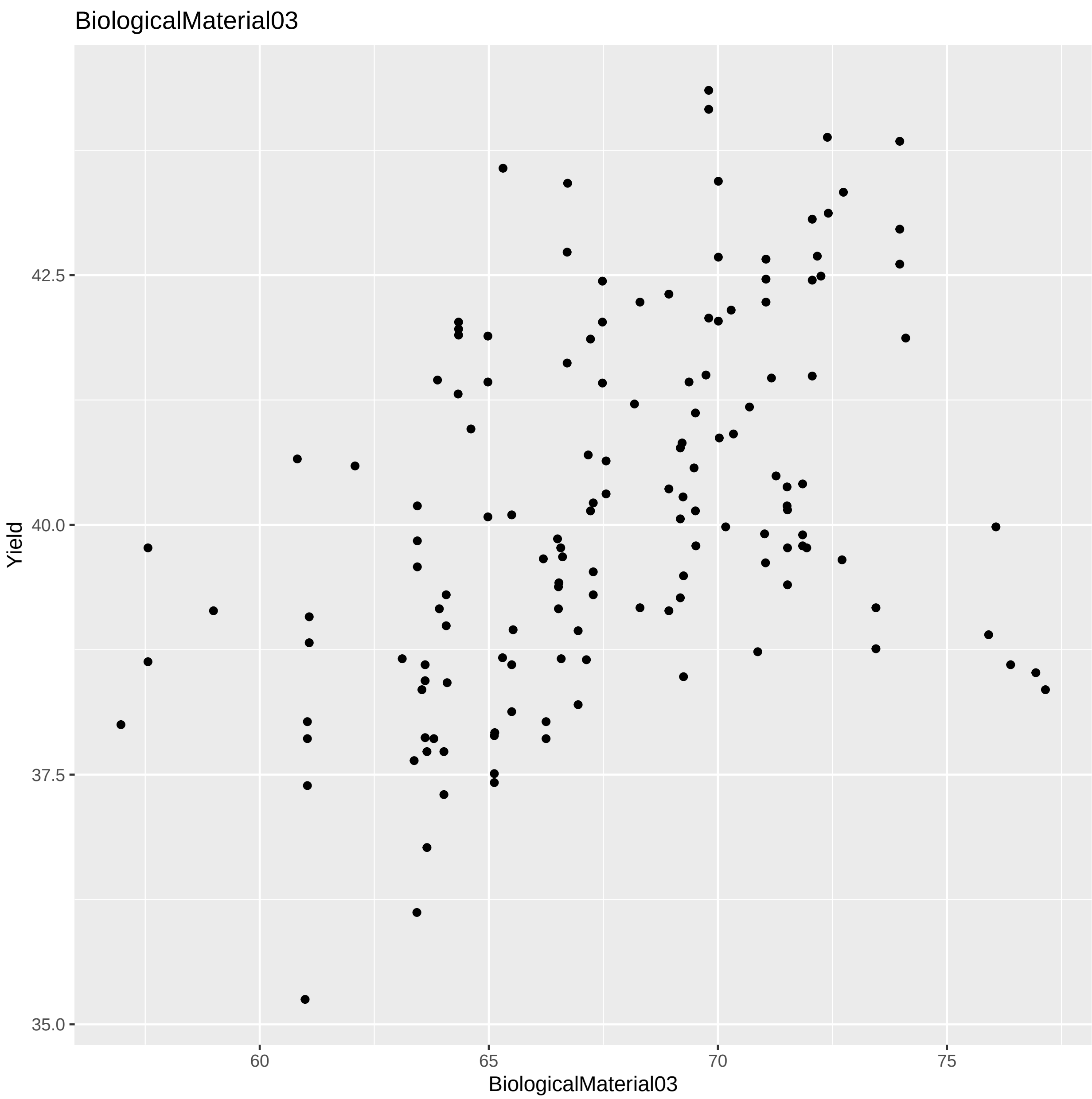
By comparing 20 varImp(nnetTune) with 20 varImp(pls_model), the former being nonlinear and latter liner, we get the following predictors as unique to nonlinear.

BiologicalMaterial03 BiologicalMaterial09 ManufacturingProcess11 ManufacturingProcess30

This was done offline manually on Bash-shell.

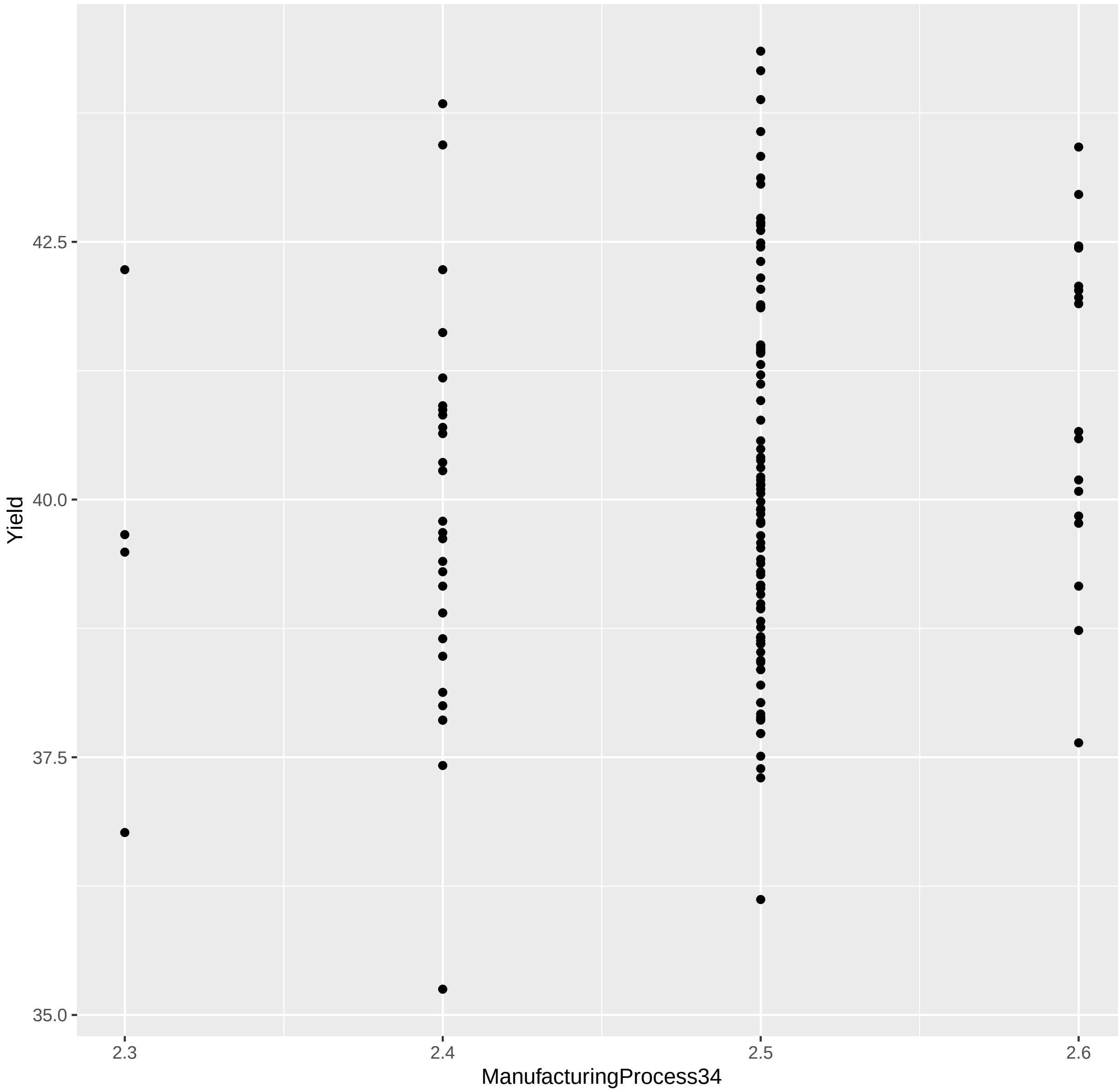
The plots of each of these variables that are unique to nonlinear set are shown below.

```r
ggplot(X_train, aes(BiologicalMaterial03, Yield)) + geom_point() + ggtitle("BiologicalMaterial03")
```

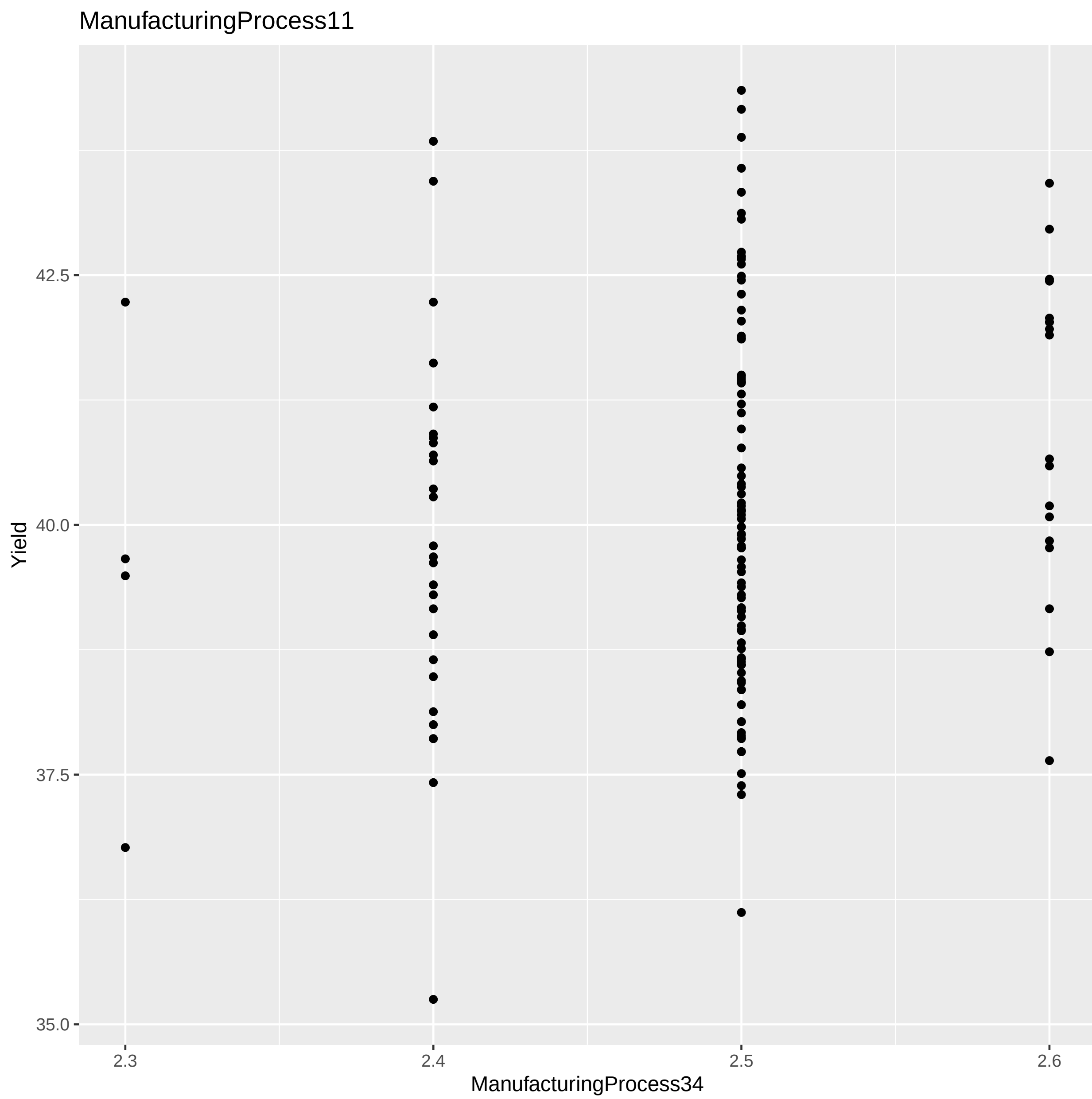



```
ggplot(X_train, aes(ManufacturingProcess34, Yield)) + geom_point() + ggtitle("BiologicalMaterial09")
```

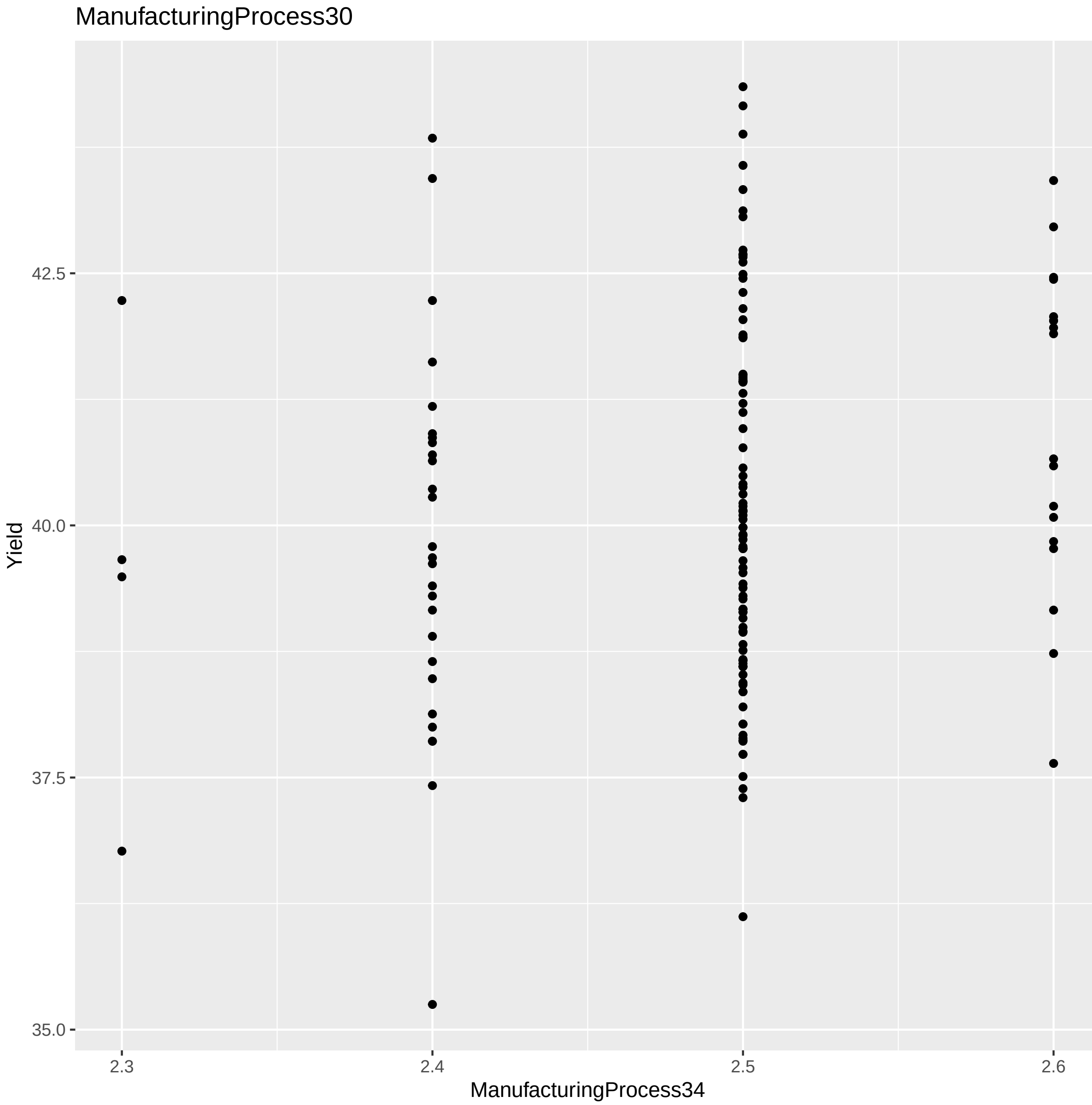
BiologicalMaterial09




```r
ggplot(X_train, aes(ManufacturingProcess34, Yield)) + geom_point() + ggtitle("ManufacturingProcess11")
```




```r
ggplot(X_train, aes(ManufacturingProcess34, Yield)) + geom_point() + ggtitle("ManufacturingProcess30")
```



These don't indicate any special relationship.