DATA624 - Homework 7

Glen Dale Davis

2023-10-31

Packages:

```
library(caret)
library(tidyverse)
library(RColorBrewer)
library(knitr)
library(pracma)
library(cowplot)
library(AppliedPredictiveModeling)
library(elasticnet)
library(glmnet)
library(VIM)
```

Exercise 6.2:

Developing a model to predict permeability (see Sect. 1.4) could save significant resources for a pharmaceutical company, while at the same time more rapidly identifying molecules that have a sufficient permeability to become a drug:

- Start R and use these commands to load the data:

```
data(permeability)
```

The matrix `fingerprints` contains the 1,107 binary molecular predictors for the 165 compounds, while `permeability` contains permeability response.

- The fingerprint predictors indicate the presence or absence of substructures of a molecule and are often sparse meaning that relatively few of the molecules contain each substructure. Filter out the predictors that have low frequencies using the `nearZeroVar` function from the `caret` package. How many predictors are left for modeling?

```
nzv_predictors <- nearZeroVar(fingerprints, names = TRUE, saveMetrics = FALSE)
fingerprints <- as.data.frame(fingerprints) |>
    select(-all_of(nzv_predictors))
print(ncol(fingerprints))
```

```
## [1] 388
```

719 near-zero-variance predictors were removed, leaving 388 predictors for modeling.

- Split the data into a training and a test set, pre-process the data, and tune a PLS model. How many latent variables are optimal and what is the corresponding resampled estimate of R^2?

We combine the predictor and response data, shuffle it, then split it into train and test sets. We then separate the predictor and response data again.

```
# Combine predictors and response
fingerprints$Permeability <- permeability

# Train and test split
set.seed(1006)
rows <- sample(nrow(fingerprints))
fingerprints <- fingerprints[rows, ]
sample <- sample(c(TRUE, FALSE), nrow(fingerprints), replace=TRUE,
                 prob=c(0.7,0.3))
train_df <- fingerprints[sample, ]
train_x <- train_df |>
    select(-Permeability)
train_y <- train_df$Permeability
train_y <- as.numeric(train_y)
test_df <- fingerprints[!sample, ]
test_x <- test_df |>
    select(-Permeability)
test_y <- test_df$Permeability
test_y <- as.numeric(test_y)
```

We check whether there are any NA values that need to be imputed.

```
# Check for NA values
any(is.na(train_df)) | any(is.na(test_df))
```

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

There are not.

We center and scale the data as a pre-processing step in tuning our model. No variables require BoxCox or other transformations. The resampling method used is 10-fold cross-validation.

```
ctrl <- trainControl(method = "cv", # 10-fold CV
                     number = 10)
# Pre-processing includes centering and scaling; no BoxCox transformations
plsTune <- train(train_x, train_y, method = "pls", tuneLength = 20,
                 trControl = ctrl, preProc = c("center", "scale"))
plsTune
```

```
## Partial Least Squares
##
## 113 samples
## 388 predictors
##
```

```
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 101, 102, 101, 101, 102, 102, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##    1     11.87895  0.3871066  8.926048
##    2     10.33501  0.5192647  7.417509
##    3     10.75715  0.4955164  8.331611
##    4     10.86789  0.4879533  8.622783
##    5     11.17420  0.4631306  8.523974
##    6     11.13900  0.4575033  8.472526
##    7     11.08195  0.4701118  8.255042
##    8     11.05509  0.4710611  8.392142
##    9     11.17666  0.4676610  8.389657
##   10     11.05191  0.4829628  8.165549
##   11     11.25075  0.4722975  8.227725
##   12     11.54057  0.4529103  8.448030
##   13     11.21358  0.4731350  8.304025
##   14     11.30998  0.4708314  8.325544
##   15     11.39502  0.4633040  8.472770
##   16     11.50098  0.4573039  8.552827
##   17     11.39542  0.4636578  8.423856
##   18     11.32989  0.4690998  8.413157
##   19     11.43207  0.4713715  8.460820
##   20     11.49542  0.4724156  8.474182
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 2.
```

The optimal number of latent variables is 2, and the corresponding resampled estimate of R^2 is 0.52.

- Predict the response for the test set. What is the test set estimate of R^2?

```
test_pred <- predict(plsTune, test_x)
SS_test_total <- sum((test_y - mean(train_y))^2)
SS_test_residual <- sum((test_y - test_pred)^2)
SS_test_regression <- sum((test_pred - mean(train_y))^2)
test_rsq <- 1 - SS_test_residual / SS_test_total
test_rsq
```

```
## [1] 0.2821783
```

```
test_rsq_check <- as.numeric(R2(test_pred, test_y, form = "traditional"))
test_rmse <- as.numeric(RMSE(test_pred, test_y))
```

We confirm our formula for predictive R^2 matches how the `R2` function from the `caret` package calculates R^2 when form is set to "traditional" by seeing if the values returned are reasonably similar.

```
round(test_rsq, 2) == round(test_rsq_check, 2)
```

```
## [1] TRUE
```

The values are reasonably similar, so we can be confident the test set estimate of R^2, i.e. predictive R^2, is 0.28.

- Try building other models discussed in this chapter. Do any have better predictive performance?

```r
ridgeGrid <- data.frame(.lambda = 10^seq(3, -3, length = 100),
                        .alpha = 0)
ridgeTune <- train(train_x, train_y, method = "glmnet",
                   tuneGrid = ridgeGrid, trControl = ctrl,
                   preProc = c("center","scale"))
ridgeTune
```

```
## glmnet
##
## 113 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 103, 101, 101, 103, 101, 102, ...
## Resampling results across tuning parameters:
##
##   lambda        RMSE      Rsquared   MAE
##   1.000000e-03  10.54902  0.5070188  7.747871
##   1.149757e-03  10.54902  0.5070188  7.747871
##   1.321941e-03  10.54902  0.5070188  7.747871
##   1.519911e-03  10.54902  0.5070188  7.747871
##   1.747528e-03  10.54902  0.5070188  7.747871
##   2.009233e-03  10.54902  0.5070188  7.747871
##   2.310130e-03  10.54902  0.5070188  7.747871
##   2.656088e-03  10.54902  0.5070188  7.747871
##   3.053856e-03  10.54902  0.5070188  7.747871
##   3.511192e-03  10.54902  0.5070188  7.747871
##   4.037017e-03  10.54902  0.5070188  7.747871
##   4.641589e-03  10.54902  0.5070188  7.747871
##   5.336699e-03  10.54902  0.5070188  7.747871
##   6.135907e-03  10.54902  0.5070188  7.747871
##   7.054802e-03  10.54902  0.5070188  7.747871
##   8.111308e-03  10.54902  0.5070188  7.747871
##   9.326033e-03  10.54902  0.5070188  7.747871
##   1.072267e-02  10.54902  0.5070188  7.747871
##   1.232847e-02  10.54902  0.5070188  7.747871
##   1.417474e-02  10.54902  0.5070188  7.747871
##   1.629751e-02  10.54902  0.5070188  7.747871
##   1.873817e-02  10.54902  0.5070188  7.747871
##   2.154435e-02  10.54902  0.5070188  7.747871
##   2.477076e-02  10.54902  0.5070188  7.747871
##   2.848036e-02  10.54902  0.5070188  7.747871
##   3.274549e-02  10.54902  0.5070188  7.747871
##   3.764936e-02  10.54902  0.5070188  7.747871
##   4.328761e-02  10.54902  0.5070188  7.747871
##   4.977024e-02  10.54902  0.5070188  7.747871
##   5.722368e-02  10.54902  0.5070188  7.747871
```

```
## 6.579332e-02 10.54902 0.5070188 7.747871
## 7.564633e-02 10.54902 0.5070188 7.747871
## 8.697490e-02 10.54902 0.5070188 7.747871
## 1.000000e-01 10.54902 0.5070188 7.747871
## 1.149757e-01 10.54902 0.5070188 7.747871
## 1.321941e-01 10.54902 0.5070188 7.747871
## 1.519911e-01 10.54902 0.5070188 7.747871
## 1.747528e-01 10.54902 0.5070188 7.747871
## 2.009233e-01 10.54902 0.5070188 7.747871
## 2.310130e-01 10.54902 0.5070188 7.747871
## 2.656088e-01 10.54902 0.5070188 7.747871
## 3.053856e-01 10.54902 0.5070188 7.747871
## 3.511192e-01 10.54902 0.5070188 7.747871
## 4.037017e-01 10.54902 0.5070188 7.747871
## 4.641589e-01 10.54902 0.5070188 7.747871
## 5.336699e-01 10.54902 0.5070188 7.747871
## 6.135907e-01 10.54902 0.5070188 7.747871
## 7.054802e-01 10.54902 0.5070188 7.747871
## 8.111308e-01 10.54902 0.5070188 7.747871
## 9.326033e-01 10.54902 0.5070188 7.747871
## 1.072267e+00 10.54902 0.5070188 7.747871
## 1.232847e+00 10.54902 0.5070188 7.747871
## 1.417474e+00 10.54902 0.5070188 7.747871
## 1.629751e+00 10.54902 0.5070188 7.747871
## 1.873817e+00 10.54902 0.5070188 7.747871
## 2.154435e+00 10.54902 0.5070188 7.747871
## 2.477076e+00 10.54902 0.5070188 7.747871
## 2.848036e+00 10.54902 0.5070188 7.747871
## 3.274549e+00 10.54902 0.5070188 7.747871
## 3.764936e+00 10.54902 0.5070188 7.747871
## 4.328761e+00 10.54902 0.5070188 7.747871
## 4.977024e+00 10.54902 0.5070188 7.747871
## 5.722368e+00 10.54902 0.5070188 7.747871
## 6.579332e+00 10.54902 0.5070188 7.747871
## 7.564633e+00 10.54902 0.5070188 7.747871
## 8.697490e+00 10.54902 0.5070188 7.747871
## 1.000000e+01 10.54902 0.5070188 7.747871
## 1.149757e+01 10.54902 0.5070188 7.747871
## 1.321941e+01 10.54902 0.5070188 7.747871
## 1.519911e+01 10.54902 0.5070188 7.747871
## 1.747528e+01 10.54902 0.5070188 7.747871
## 2.009233e+01 10.54902 0.5070188 7.747871
## 2.310130e+01 10.54902 0.5070188 7.747871
## 2.656088e+01 10.54902 0.5070188 7.747871
## 3.053856e+01 10.54902 0.5070188 7.747871
## 3.511192e+01 10.54902 0.5070188 7.747871
## 4.037017e+01 10.54902 0.5070188 7.747871
## 4.641589e+01 10.54902 0.5070188 7.747871
## 5.336699e+01 10.54902 0.5070188 7.747871
## 6.135907e+01 10.54902 0.5070188 7.747871
## 7.054802e+01 10.54902 0.5070188 7.747871
## 8.111308e+01 10.54902 0.5070188 7.747871
## 9.326033e+01 10.56218 0.5068871 7.758079
## 1.072267e+02 10.62331 0.5065411 7.802072
```

```
##   1.232847e+02  10.68646  0.5043457  7.866862
##   1.417474e+02  10.75769  0.5018240  7.950966
##   1.629751e+02  10.83729  0.4990086  8.038374
##   1.873817e+02  10.92565  0.4958817  8.137249
##   2.154435e+02  11.02297  0.4924276  8.243812
##   2.477076e+02  11.12914  0.4886494  8.359497
##   2.848036e+02  11.24480  0.4844948  8.488132
##   3.274549e+02  11.36885  0.4800048  8.628686
##   3.764936e+02  11.50071  0.4751696  8.770239
##   4.328761e+02  11.63951  0.4699963  8.912036
##   4.977024e+02  11.78414  0.4645013  9.053297
##   5.722368e+02  11.93328  0.4587093  9.196039
##   6.579332e+02  12.08557  0.4526222  9.340859
##   7.564633e+02  12.23951  0.4463633  9.487044
##   8.697490e+02  12.39355  0.4399504  9.642277
##   1.000000e+03  12.54587  0.4334778  9.791916
##
## Tuning parameter 'alpha' was held constant at a value of 0
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0 and lambda = 81.11308.
```

```r
test_pred2 <- predict(ridgeTune, test_x)
test_rsq2 <- as.numeric(R2(test_pred2, test_y, form = "traditional"))
test_rmse2 <- as.numeric(RMSE(test_pred2, test_y))
test_rsq2
```

```
## [1] 0.3697419
```

A ridge regression model with $\lambda = 81.11308$ has a higher predictive R^2 than the PLS model, but it also has a slightly higher $RMSE$.

```r
lassoGrid <- ridgeGrid |>
    mutate(.alpha = 1)
lassoTune <- train(train_x, train_y, method = "glmnet",
                   tuneGrid = lassoGrid, trControl = ctrl,
                   preProc = c("center","scale"))
lassoTune
```

```
## glmnet
##
## 113 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 101, 101, 103, 101, 103, 103, ...
## Resampling results across tuning parameters:
##
##   lambda        RMSE      Rsquared   MAE
##   1.000000e-03  11.00010  0.5502208   8.228665
##   1.149757e-03  11.00010  0.5502208   8.228665
##   1.321941e-03  11.00010  0.5502208   8.228665
##   1.519911e-03  11.00010  0.5502208   8.228665
```

```
## 1.747528e-03 11.00010 0.5502208 8.228665
## 2.009233e-03 11.00010 0.5502208 8.228665
## 2.310130e-03 11.00010 0.5502208 8.228665
## 2.656088e-03 11.00010 0.5502208 8.228665
## 3.053856e-03 11.00010 0.5502208 8.228665
## 3.511192e-03 11.00010 0.5502208 8.228665
## 4.037017e-03 11.00010 0.5502208 8.228665
## 4.641589e-03 11.00010 0.5502208 8.228665
## 5.336699e-03 11.00010 0.5502208 8.228665
## 6.135907e-03 11.00010 0.5502208 8.228665
## 7.054802e-03 11.00010 0.5502208 8.228665
## 8.111308e-03 11.00010 0.5502208 8.228665
## 9.326033e-03 11.00010 0.5502208 8.228665
## 1.072267e-02 11.00010 0.5502208 8.228665
## 1.232847e-02 11.00010 0.5502208 8.228665
## 1.417474e-02 11.00010 0.5502208 8.228665
## 1.629751e-02 11.00010 0.5502208 8.228665
## 1.873817e-02 11.00010 0.5502208 8.228665
## 2.154435e-02 11.00010 0.5502208 8.228665
## 2.477076e-02 11.00010 0.5502208 8.228665
## 2.848036e-02 11.00010 0.5502208 8.228665
## 3.274549e-02 11.00010 0.5502208 8.228665
## 3.764936e-02 11.00010 0.5502208 8.228665
## 4.328761e-02 11.00010 0.5502208 8.228665
## 4.977024e-02 11.00010 0.5502208 8.228665
## 5.722368e-02 11.00010 0.5502208 8.228665
## 6.579332e-02 11.00010 0.5502208 8.228665
## 7.564633e-02 11.00010 0.5502208 8.228665
## 8.697490e-02 11.00010 0.5502208 8.228665
## 1.000000e-01 10.99403 0.5512474 8.223282
## 1.149757e-01 10.95302 0.5571591 8.148766
## 1.321941e-01 10.92517 0.5620144 8.119638
## 1.519911e-01 10.87108 0.5690207 8.045694
## 1.747528e-01 10.76291 0.5785965 7.916859
## 2.009233e-01 10.62743 0.5858136 7.784030
## 2.310130e-01 10.51124 0.5883109 7.698825
## 2.656088e-01 10.48818 0.5841775 7.683142
## 3.053856e-01 10.51361 0.5779310 7.716632
## 3.511192e-01 10.63409 0.5637939 7.830234
## 4.037017e-01 10.79128 0.5488431 7.963783
## 4.641589e-01 10.99437 0.5306196 8.128397
## 5.336699e-01 11.24202 0.5080484 8.368544
## 6.135907e-01 11.51378 0.4819259 8.600872
## 7.054802e-01 11.70291 0.4606707 8.758681
## 8.111308e-01 11.84195 0.4456170 8.847260
## 9.326033e-01 11.82014 0.4442721 8.806428
## 1.072267e+00 11.73760 0.4492059 8.714355
## 1.232847e+00 11.65504 0.4562766 8.639493
## 1.417474e+00 11.61721 0.4607032 8.634578
## 1.629751e+00 11.67641 0.4584730 8.723026
## 1.873817e+00 11.76784 0.4535561 8.834898
## 2.154435e+00 11.81475 0.4553939 8.904093
## 2.477076e+00 11.82889 0.4640392 8.975429
## 2.848036e+00 11.84471 0.4748238 9.045688
```

```
##   3.274549e+00  11.90506  0.4834221   9.173139
##   3.764936e+00  11.99404  0.4893104   9.300832
##   4.328761e+00  12.13459  0.4934943   9.469261
##   4.977024e+00  12.33639  0.4994432   9.695546
##   5.722368e+00  12.62322  0.5027345   9.963476
##   6.579332e+00  13.00725  0.5042864  10.293518
##   7.564633e+00  13.49577  0.5076145  10.703860
##   8.697490e+00  14.13121  0.5093814  11.221745
##   1.000000e+01  14.70948  0.2363377  11.662931
##   1.149757e+01  14.74279        NaN  11.684565
##   1.321941e+01  14.74279        NaN  11.684565
##   1.519911e+01  14.74279        NaN  11.684565
##   1.747528e+01  14.74279        NaN  11.684565
##   2.009233e+01  14.74279        NaN  11.684565
##   2.310130e+01  14.74279        NaN  11.684565
##   2.656088e+01  14.74279        NaN  11.684565
##   3.053856e+01  14.74279        NaN  11.684565
##   3.511192e+01  14.74279        NaN  11.684565
##   4.037017e+01  14.74279        NaN  11.684565
##   4.641589e+01  14.74279        NaN  11.684565
##   5.336699e+01  14.74279        NaN  11.684565
##   6.135907e+01  14.74279        NaN  11.684565
##   7.054802e+01  14.74279        NaN  11.684565
##   8.111308e+01  14.74279        NaN  11.684565
##   9.326033e+01  14.74279        NaN  11.684565
##   1.072267e+02  14.74279        NaN  11.684565
##   1.232847e+02  14.74279        NaN  11.684565
##   1.417474e+02  14.74279        NaN  11.684565
##   1.629751e+02  14.74279        NaN  11.684565
##   1.873817e+02  14.74279        NaN  11.684565
##   2.154435e+02  14.74279        NaN  11.684565
##   2.477076e+02  14.74279        NaN  11.684565
##   2.848036e+02  14.74279        NaN  11.684565
##   3.274549e+02  14.74279        NaN  11.684565
##   3.764936e+02  14.74279        NaN  11.684565
##   4.328761e+02  14.74279        NaN  11.684565
##   4.977024e+02  14.74279        NaN  11.684565
##   5.722368e+02  14.74279        NaN  11.684565
##   6.579332e+02  14.74279        NaN  11.684565
##   7.564633e+02  14.74279        NaN  11.684565
##   8.697490e+02  14.74279        NaN  11.684565
##   1.000000e+03  14.74279        NaN  11.684565
##
## Tuning parameter 'alpha' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.2656088.
```

```r
test_pred3 <- predict(lassoTune, test_x)
test_rsq3 <- as.numeric(R2(test_pred3, test_y, form = "traditional"))
test_rmse3 <- as.numeric(RMSE(test_pred3, test_y))
test_rsq3
```

```
## [1] 0.2918282
```

A lasso regression model with $\lambda = 0.2656088$ has a slightly higher predictive R^2 than the PLS model, but it also has a slightly higher $RMSE$.

We create a summary table where it's easier to see and compare the various resample $RMSE$ metrics.

```
models <- list(pls = plsTune, ridge = ridgeTune, lasso = lassoTune)
resamples(models) |> summary(metric = "RMSE")
```

```
##
## Call:
## summary.resamples(object = resamples(models), metric = "RMSE")
##
## Models: pls, ridge, lasso
## Number of resamples: 10
##
## RMSE
##           Min.  1st Qu.    Median     Mean  3rd Qu.     Max. NA's
## pls   5.834145 7.810709  9.938699 10.33501 12.17152 17.31642    0
## ridge 8.015161 9.079117 10.311265 10.54902 11.13044 16.20453    0
## lasso 6.293448 8.423313 10.615106 10.48818 11.89339 14.81206    0
```

We conclude the PLS model should be a better predictor than either the ridge or lasso regression models. If we were to compare test set $RMSE$ instead, the ridge regression model scores the best, however:

```
tbl <- data.frame(model = c("pls", "ridge", "lasso"),
                  test_set_RMSE = c(test_rmse, test_rmse2, test_rmse3))
knitr::kable(tbl, format = "simple")
```

model	test_set_RMSE
pls	14.31040
ridge	13.40191
lasso	14.20616

- Would you recommend any of your models to replace the permeability laboratory experiment?

Despite the fact that the experiment is expensive, there would also be costs associated with making incorrect predictions. Without knowing what either costs are, we can't analyze the trade-offs, so no.

Exercise 6.3:

A chemical manufacturing process for a pharmaceutical product was discussed in Sect. 1.4. In this problem, the objective is to understand the relationship between biological measurements of the raw materials (predictors), measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1% will boost revenue by approximately one hundred thousand dollars per batch:

- Start R and use these commands to load the data:

```r
data(ChemicalManufacturingProcess)
```

The matrix `processPredictors` contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. `yield` contains the percent yield for each run.

- A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values (e.g., see Sect. 3.8).

```r
x <- colSums(is.na(ChemicalManufacturingProcess))
missing_val_cols <- names(x[x > 0])
ChemicalManufacturingProcess <- ChemicalManufacturingProcess |>
    VIM::kNN(variable = missing_val_cols, k = 15, numFun = weighted.mean,
             weightDist = TRUE, imp_var = FALSE)
```

- Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter. What is the optimal value of the performance metric?

```r
nzv_predictors <- nearZeroVar(ChemicalManufacturingProcess |> select(-Yield),
                              names = TRUE, saveMetrics = FALSE)
ChemicalManufacturingProcess <- ChemicalManufacturingProcess |>
    select(-all_of(nzv_predictors))
rows <- sample(nrow(ChemicalManufacturingProcess))
ChemicalManufacturingProcess <- ChemicalManufacturingProcess[rows, ]
sample <- sample(c(TRUE, FALSE), nrow(ChemicalManufacturingProcess),
                 replace=TRUE, prob=c(0.7,0.3))
train_df2 <- ChemicalManufacturingProcess[sample, ]
train_x2 <- train_df2 |>
    select(-Yield)
train_y2 <- train_df2$Yield
train_y2 <- as.numeric(train_y2)
test_df2 <- ChemicalManufacturingProcess[!sample, ]
test_x2 <- test_df2 |>
    select(-Yield)
test_y2 <- test_df2$Yield
test_y2 <- as.numeric(test_y2)
lassoTune2 <- train(train_x2, train_y2, method = "glmnet",
                    tuneGrid = lassoGrid, trControl = ctrl,
                    preProc = c("center","scale"))
lassoTune2
```

```
## glmnet
##
## 128 samples
##  56 predictor
##
## Pre-processing: centered (56), scaled (56)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 115, 116, 116, 115, 114, 116, ...
## Resampling results across tuning parameters:
##
##   lambda        RMSE      Rsquared   MAE
```

```
##   1.000000e-03  3.325662  0.4519000  1.6803597
##   1.149757e-03  3.297578  0.4558300  1.6689548
##   1.321941e-03  3.258774  0.4601782  1.6544365
##   1.519911e-03  3.236074  0.4642824  1.6444887
##   1.747528e-03  3.128114  0.4683810  1.6083939
##   2.009233e-03  2.991677  0.4739989  1.5643270
##   2.310130e-03  2.869064  0.4807068  1.5249845
##   2.656088e-03  2.754208  0.4863737  1.4897245
##   3.053856e-03  2.642794  0.4913548  1.4557466
##   3.511192e-03  2.523845  0.4974022  1.4168232
##   4.037017e-03  2.387964  0.5051406  1.3718510
##   4.641589e-03  2.253694  0.5136632  1.3276391
##   5.336699e-03  2.123991  0.5229760  1.2831600
##   6.135907e-03  1.999270  0.5335993  1.2406576
##   7.054802e-03  1.861699  0.5466180  1.1926773
##   8.111308e-03  1.729307  0.5612375  1.1454988
##   9.326033e-03  1.579942  0.5801392  1.0876310
##   1.072267e-02  1.455844  0.5940605  1.0300989
##   1.232847e-02  1.350948  0.5943242  1.0069764
##   1.417474e-02  1.318165  0.5930761  0.9996385
##   1.629751e-02  1.286420  0.5999407  0.9901878
##   1.873817e-02  1.272179  0.6207881  0.9773197
##   2.154435e-02  1.284234  0.6269569  0.9710604
##   2.477076e-02  1.314067  0.6205520  0.9847900
##   2.848036e-02  1.345953  0.6140280  0.9997968
##   3.274549e-02  1.360874  0.6110979  1.0083246
##   3.764936e-02  1.324069  0.6137491  0.9979382
##   4.328761e-02  1.219356  0.6251917  0.9643920
##   4.977024e-02  1.127317  0.6497920  0.9238382
##   5.722368e-02  1.116216  0.6572931  0.9125928
##   6.579332e-02  1.119111  0.6567836  0.9153795
##   7.564633e-02  1.125343  0.6537918  0.9192235
##   8.697490e-02  1.131352  0.6504996  0.9224115
##   1.000000e-01  1.138437  0.6473802  0.9247277
##   1.149757e-01  1.144539  0.6454215  0.9260769
##   1.321941e-01  1.144092  0.6474803  0.9253231
##   1.519911e-01  1.144444  0.6501207  0.9270678
##   1.747528e-01  1.152034  0.6490403  0.9360841
##   2.009233e-01  1.161032  0.6489251  0.9430269
##   2.310130e-01  1.173608  0.6471601  0.9571629
##   2.656088e-01  1.191280  0.6438868  0.9746401
##   3.053856e-01  1.214148  0.6393962  0.9954535
##   3.511192e-01  1.239402  0.6355972  1.0187356
##   4.037017e-01  1.269421  0.6310851  1.0480949
##   4.641589e-01  1.305171  0.6310124  1.0775877
##   5.336699e-01  1.351011  0.6320395  1.1162360
##   6.135907e-01  1.410637  0.6324553  1.1647883
##   7.054802e-01  1.487119  0.6320886  1.2259384
##   8.111308e-01  1.584254  0.6274204  1.3028101
##   9.326033e-01  1.704528  0.6017249  1.3991884
##   1.072267e+00  1.828560  0.4000238  1.4990603
##   1.232847e+00  1.842860        NaN  1.5094636
##   1.417474e+00  1.842860        NaN  1.5094636
##   1.629751e+00  1.842860        NaN  1.5094636
```

```
##   1.873817e+00  1.842860        NaN  1.5094636
##   2.154435e+00  1.842860        NaN  1.5094636
##   2.477076e+00  1.842860        NaN  1.5094636
##   2.848036e+00  1.842860        NaN  1.5094636
##   3.274549e+00  1.842860        NaN  1.5094636
##   3.764936e+00  1.842860        NaN  1.5094636
##   4.328761e+00  1.842860        NaN  1.5094636
##   4.977024e+00  1.842860        NaN  1.5094636
##   5.722368e+00  1.842860        NaN  1.5094636
##   6.579332e+00  1.842860        NaN  1.5094636
##   7.564633e+00  1.842860        NaN  1.5094636
##   8.697490e+00  1.842860        NaN  1.5094636
##   1.000000e+01  1.842860        NaN  1.5094636
##   1.149757e+01  1.842860        NaN  1.5094636
##   1.321941e+01  1.842860        NaN  1.5094636
##   1.519911e+01  1.842860        NaN  1.5094636
##   1.747528e+01  1.842860        NaN  1.5094636
##   2.009233e+01  1.842860        NaN  1.5094636
##   2.310130e+01  1.842860        NaN  1.5094636
##   2.656088e+01  1.842860        NaN  1.5094636
##   3.053856e+01  1.842860        NaN  1.5094636
##   3.511192e+01  1.842860        NaN  1.5094636
##   4.037017e+01  1.842860        NaN  1.5094636
##   4.641589e+01  1.842860        NaN  1.5094636
##   5.336699e+01  1.842860        NaN  1.5094636
##   6.135907e+01  1.842860        NaN  1.5094636
##   7.054802e+01  1.842860        NaN  1.5094636
##   8.111308e+01  1.842860        NaN  1.5094636
##   9.326033e+01  1.842860        NaN  1.5094636
##   1.072267e+02  1.842860        NaN  1.5094636
##   1.232847e+02  1.842860        NaN  1.5094636
##   1.417474e+02  1.842860        NaN  1.5094636
##   1.629751e+02  1.842860        NaN  1.5094636
##   1.873817e+02  1.842860        NaN  1.5094636
##   2.154435e+02  1.842860        NaN  1.5094636
##   2.477076e+02  1.842860        NaN  1.5094636
##   2.848036e+02  1.842860        NaN  1.5094636
##   3.274549e+02  1.842860        NaN  1.5094636
##   3.764936e+02  1.842860        NaN  1.5094636
##   4.328761e+02  1.842860        NaN  1.5094636
##   4.977024e+02  1.842860        NaN  1.5094636
##   5.722368e+02  1.842860        NaN  1.5094636
##   6.579332e+02  1.842860        NaN  1.5094636
##   7.564633e+02  1.842860        NaN  1.5094636
##   8.697490e+02  1.842860        NaN  1.5094636
##   1.000000e+03  1.842860        NaN  1.5094636
##
## Tuning parameter 'alpha' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.05722368.
```

A lasso model with $\lambda = 0.05722368$ has the smallest $RMSE$ at 1.116216.

- Predict the response for the test set. What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

```
test_pred4 <- predict(lassoTune2, test_x2)
test_rsq4 <- as.numeric(R2(test_pred4, test_y2, form = "traditional"))
test_rmse4 <- as.numeric(RMSE(test_pred4, test_y2))
tbl <- data.frame(model = c("lasso", "lasso"),
                  metric = c("test_set_Rsq", "test_set_RMSE"),
                  value = c(test_rsq4, test_rmse4))
knitr::kable(tbl, format = "simple")
```

model	metric	value
lasso	test_set_Rsq	0.3884768
lasso	test_set_RMSE	1.4111490

The value of the test set $RMSE$ is higher at 1.4111490.

- Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

The 20 most important predictors in the model we've trained are:

```
var_imp <- varImp(lassoTune2, lambda = lassoTune2$lambda.min)
var_imp <- var_imp$importance |>
    arrange(desc(Overall)) |>
    top_n(20) |>
    rownames_to_column()
cols <- c("Predictor", "Importance")
colnames(var_imp) <- cols
knitr::kable(var_imp, format = "simple")
```

Predictor	Importance
ManufacturingProcess09	100.000000
ManufacturingProcess32	98.875327
ManufacturingProcess34	39.274706
ManufacturingProcess45	26.743478
ManufacturingProcess37	23.741268
ManufacturingProcess17	22.028702
ManufacturingProcess29	21.975921
ManufacturingProcess28	18.885247
ManufacturingProcess36	18.656307
BiologicalMaterial05	18.097788
ManufacturingProcess07	17.066103
BiologicalMaterial03	15.510297
ManufacturingProcess06	15.046292
ManufacturingProcess04	12.151953
ManufacturingProcess13	7.912206
ManufacturingProcess01	6.894576
ManufacturingProcess43	6.122423
ManufacturingProcess15	4.891769

Predictor	Importance
ManufacturingProcess30	3.863499
ManufacturingProcess41	3.527710

```
var_imp_names <- var_imp$Predictor
```

Out of these top 20 predictors, the manufacturing process variables dominate. Only two biological material variables are in the top 20.

- Explore the relationships between each of the top predictors and the response. How could this information be helpful in improving yield in future runs of the manufacturing process?

The coefficients for the top 20 predictors are:

```
coef_var_imp <- as.matrix(coef(lassoTune2$finalModel, lassoTune2$bestTune$lambda))
coef_var_imp <- as.data.frame(coef_var_imp) |>
    rownames_to_column() |>
    filter(rowname %in% var_imp_names) |>
    arrange(desc(s1))
cols <- c("Predictor", "Coefficient")
colnames(coef_var_imp) <- cols
knitr::kable(coef_var_imp, format = "simple")
```

Predictor	Coefficient
ManufacturingProcess09	0.7358340
ManufacturingProcess32	0.7275583
ManufacturingProcess34	0.2889967
ManufacturingProcess45	0.1967876
ManufacturingProcess29	0.1617063
BiologicalMaterial05	0.1331697
BiologicalMaterial03	0.1141300
ManufacturingProcess06	0.1107157
ManufacturingProcess04	0.0894182
ManufacturingProcess01	0.0507326
ManufacturingProcess43	0.0450509
ManufacturingProcess15	0.0359953
ManufacturingProcess30	0.0284289
ManufacturingProcess41	-0.0259581
ManufacturingProcess13	-0.0582207
ManufacturingProcess07	-0.1255782
ManufacturingProcess36	-0.1372795
ManufacturingProcess28	-0.1389641
ManufacturingProcess17	-0.1620947
ManufacturingProcess37	-0.1746963

To improve yield, these are the variables that we need to increase (in order of importance):

```r
coef_pos_impact <- coef_var_imp |>
    filter(Coefficient > 0)
knitr::kable(coef_pos_impact, format = "simple")
```

Predictor	Coefficient
ManufacturingProcess09	0.7358340
ManufacturingProcess32	0.7275583
ManufacturingProcess34	0.2889967
ManufacturingProcess45	0.1967876
ManufacturingProcess29	0.1617063
BiologicalMaterial05	0.1331697
BiologicalMaterial03	0.1141300
ManufacturingProcess06	0.1107157
ManufacturingProcess04	0.0894182
ManufacturingProcess01	0.0507326
ManufacturingProcess43	0.0450509
ManufacturingProcess15	0.0359953
ManufacturingProcess30	0.0284289

And these are the variables that we need to decrease (also in order of importance):

```r
coef_neg_impact <- coef_var_imp |>
    filter(Coefficient < 0) |>
    arrange(Coefficient)
knitr::kable(coef_neg_impact, format = "simple")
```

Predictor	Coefficient
ManufacturingProcess37	-0.1746963
ManufacturingProcess17	-0.1620947
ManufacturingProcess28	-0.1389641
ManufacturingProcess36	-0.1372795
ManufacturingProcess07	-0.1255782
ManufacturingProcess13	-0.0582207
ManufacturingProcess41	-0.0259581