Data624 - Homework7

Amit Kapoor

3/28/2021

Contents

```
library(AppliedPredictiveModeling)
library(tidyverse)
library(caret)
library(naniar)
library(corrplot)
```

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:

(a)

Start R and use these commands to load the data.

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

```
data(permeability)
```

```
glimpse(permeability)
```

```
##  num [1:165, 1] 12.52 1.12 19.41 1.73 1.68 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:165] "1" "2" "3" "4" ...
##   ..$ : chr "permeability"
```

```
nrow(permeability)
```

```
## [1] 165
```

```
glimpse(fingerprints)
```

```
##  num [1:165, 1:1107] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:165] "1" "2" "3" "4" ...
##   ..$ : chr [1:1107] "X1" "X2" "X3" "X4" ...
```

```
ncol(fingerprints)
```

```
## [1] 1107
```

```
nrow(fingerprints)
```

```
## [1] 165
```

In this data there were 165 unique compounds; 1107 molecular fingerprints were determined for each. A molecular fingerprint is a binary sequence of numbers that represents the presence or absence of a specific molecular sub-structure. The response is highly skewed, the predictors are sparse (15.5 percent are present), and many predictors are strongly associated

(b)

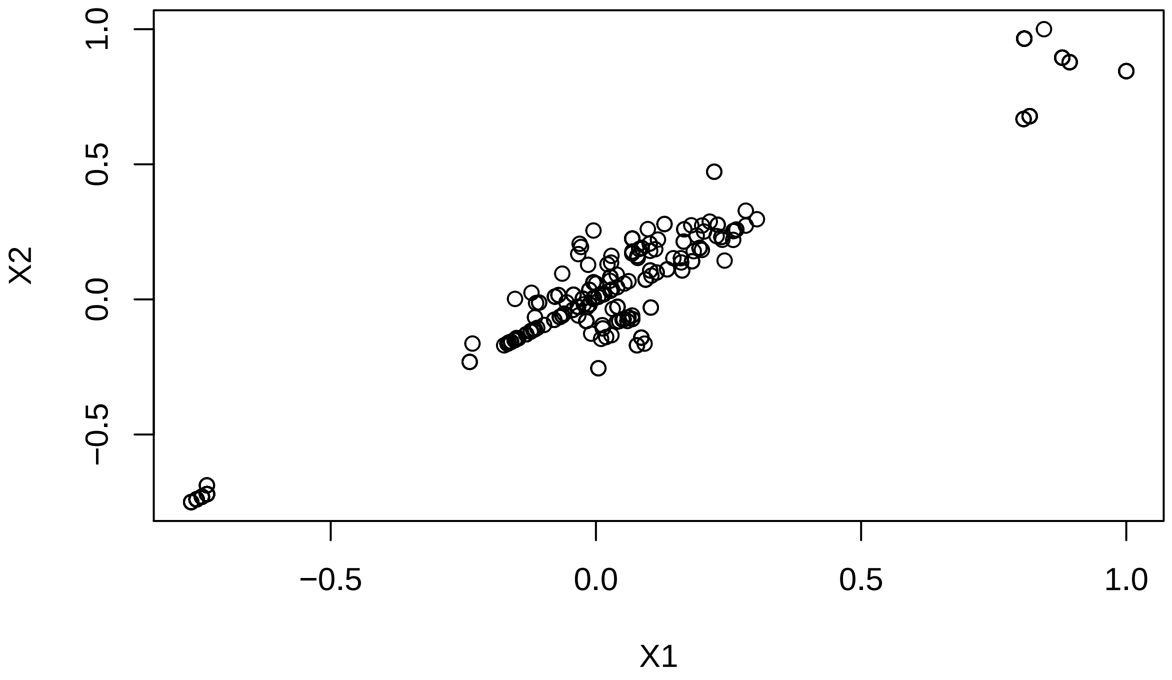
The fingerprint predictors indicate the presence or absence of substructures of a molecule and are often sparse meaning that relatively few of the molecules contain each substructure. Filter out the predictors that have low frequencies using the `nearZeroVar` function from the caret package. How many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

```
remove.features <- nearZeroVar(fingerprints)
X <- fingerprints[,-remove.features]
length(remove.features) %>% paste('columns are removed. ', dim(X)[2], ' predictors are left for modelin
```

```
## [1] "719 columns are removed.  388  predictors are left for modeling."
```

We will now look into pairwise correlation above 0.90. We will then remove the predictors having correlation with cutoff 0.90.

```
plot(cor(X))
```



```
corr.90 <- findCorrelation(cor(X), cutoff=0.90)
X <- X[,-corr.90]
length(corr.90) %>% paste('columns having correlation 0.90 or more are removed. ', dim(X)[2], ' predicto
```

```
## [1] "278 columns having correlation 0.90 or more are removed.  110  predictors are left for modeling
```

(c)

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?

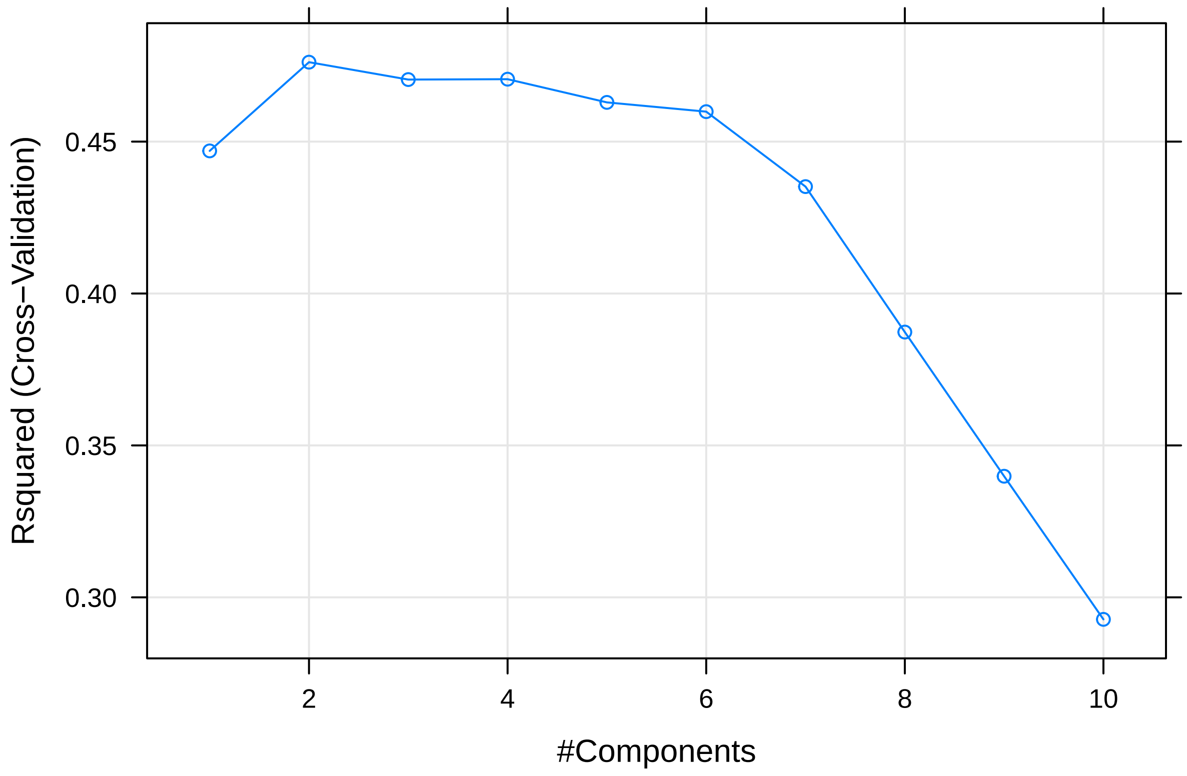
We will do the train and test set partition as 70% and 30% respectively and then fit the pls model using R^2 as the metric. The train function uses `center` and `scale` where center will subtract the mean of predictor from corresponding value and scale to divide by sd.

```
set.seed(786)
partition <- createDataPartition(permeability, p=0.70, list = FALSE)

# predictor
X.train <- X[partition,]
X.test <- X[-partition,]

# response
y.train <- permeability[partition,]
y.test <- permeability[-partition,]

# tune pls model
pls.fit <- train(x=X.train,
                 y=y.train,
                 method="pls",
                 metric="Rsquared",
                 tuneLength=10,
                 trControl=trainControl(method = "cv"),
                 preProcess=c("center", "scale")
```

```
                  )

pls.fit
```

```
## Partial Least Squares
##
## 117 samples
## 110 predictors
##
## Pre-processing: centered (110), scaled (110)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 107, 105, 105, 106, 105, 105, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##    1     11.63701  0.4469410   8.212073
##    2     11.19413  0.4761420   8.071552
##    3     11.33762  0.4704264   8.606355
##    4     11.37860  0.4705414   8.654649
##    5     11.49895  0.4629129   8.619412
##    6     11.77805  0.4598600   8.916459
##    7     12.31715  0.4351858   9.354903
##    8     13.33637  0.3873041   9.915650
##    9     13.96776  0.3398617  10.369458
##   10     14.81892  0.2927483  10.927883
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 2.
```

```
# plot R-squared vs components
plot(pls.fit)
```



```
pls.fit$results %>%
  filter(ncomp == pls.fit$bestTune$ncomp) %>%
  select(ncomp,RMSE,Rsquared)
```

```
##   ncomp     RMSE Rsquared
## 1     2 11.19413 0.476142
```

After applying partial least square model, we see now that number of components 2 produces minimum RMSE (11.19413) and max R^2 (0.476142).

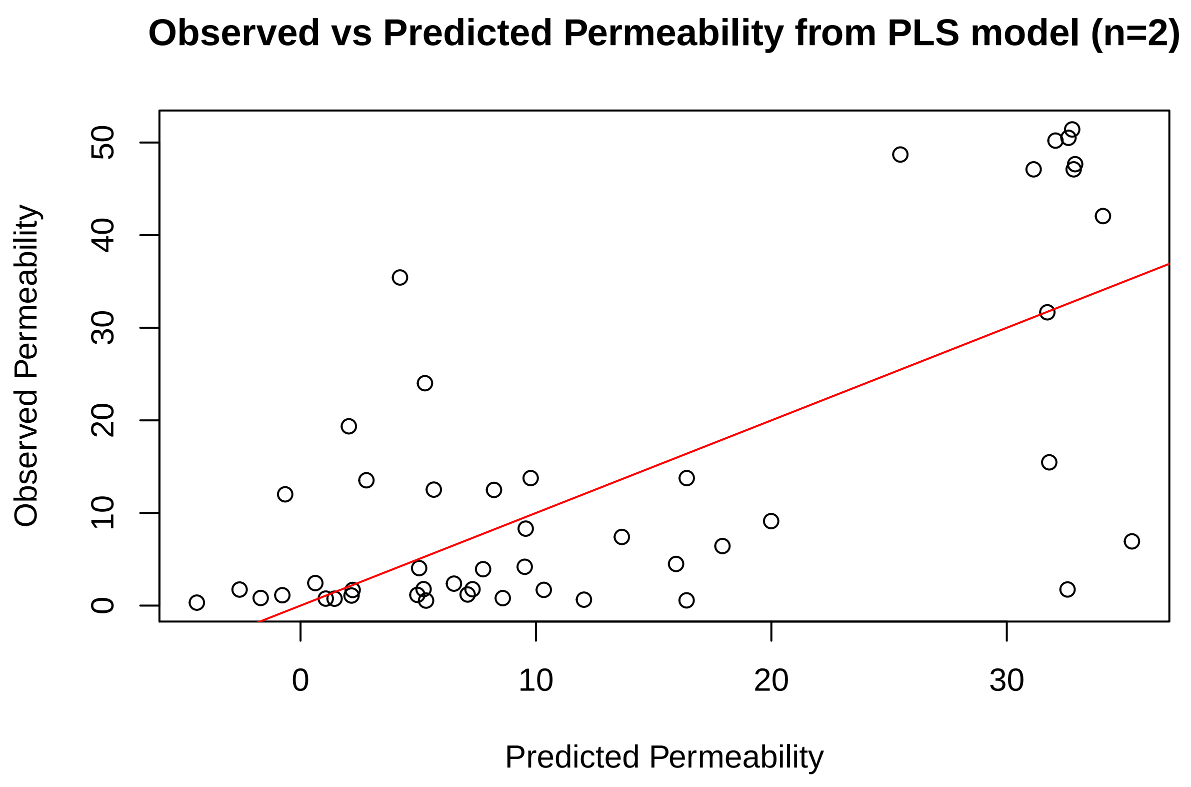
(d)

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

```
# Prediction from pls model
pls.prediction <- predict(pls.fit, X.test)
postResample(pred=pls.prediction, obs=y.test)
```

```
##       RMSE   Rsquared        MAE
## 12.4644848  0.4761679  9.4899066
```

```
# plot
plot(pls.prediction, y.test,
     main="Observed vs Predicted Permeability from PLS model (n=2)",
     xlab="Predicted Permeability",
     ylab="Observed Permeability")
abline(0,1,col="red")
```



The plot shows the predicted vs observed permeability. The performance shows that the model fits the test data with RMSE as 12.464 and handles 48% of variability of data.

(e)

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

We will try to fit below 3 penalized models and compare the performance against the PLS model:

- Ridge regression: param tuned would be λ (0 to 1 by 0.1)
- Lasso regression: param tuned would be fraction (0 to 0.5 by 0.05)
- Elastic net: param tuned would be fraction and λ (0 to 1 by 0.1)

We will set the same seed for all the model. R^2 matrix will be used for all the models evaluation.

```
set.seed(786)
# tune ridge model
ridge.fit <- train(x=X.train,
                 y=y.train,
                 method="ridge",
                 metric="Rsquared",
                 tuneGrid = data.frame(lambda=seq(0,1,by=0.1)),
                 trControl=trainControl(method = "cv",number=10),
                 preProcess=c("center", "scale")
                 )

ridge.fit
```

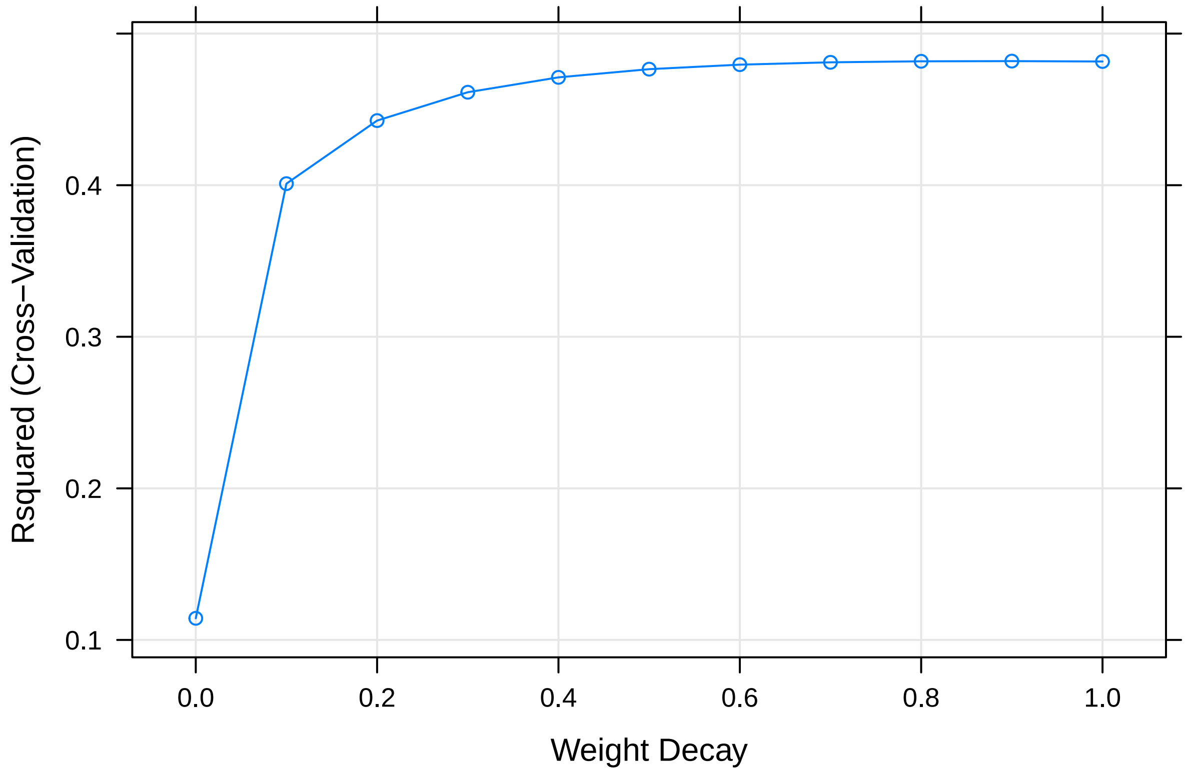
```
## Ridge Regression
##
## 117 samples
## 110 predictors
```

```
##
## Pre-processing: centered (110), scaled (110)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 106, 105, 105, 105, 106, 106, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE          Rsquared   MAE
##   0.0     1.357937e+15  0.1142670  3.920028e+14
##   0.1     1.307559e+01  0.4010144  9.842309e+00
##   0.2     1.252247e+01  0.4426120  9.395175e+00
##   0.3     1.242482e+01  0.4613298  9.289079e+00
##   0.4     1.250699e+01  0.4711338  9.342353e+00
##   0.5     1.269048e+01  0.4765397  9.538792e+00
##   0.6     1.294065e+01  0.4795180  9.798881e+00
##   0.7     1.323834e+01  0.4810652  1.009804e+01
##   0.8     1.357141e+01  0.4817329  1.041612e+01
##   0.9     1.393148e+01  0.4818460  1.074867e+01
##   1.0     1.431243e+01  0.4816042  1.107811e+01
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was lambda = 0.9.
```

```r
ridge.fit$bestTune
```

```
##    lambda
## 10    0.9
```

```r
plot(ridge.fit)
```



```r
set.seed(786)
# tune lasso model
```

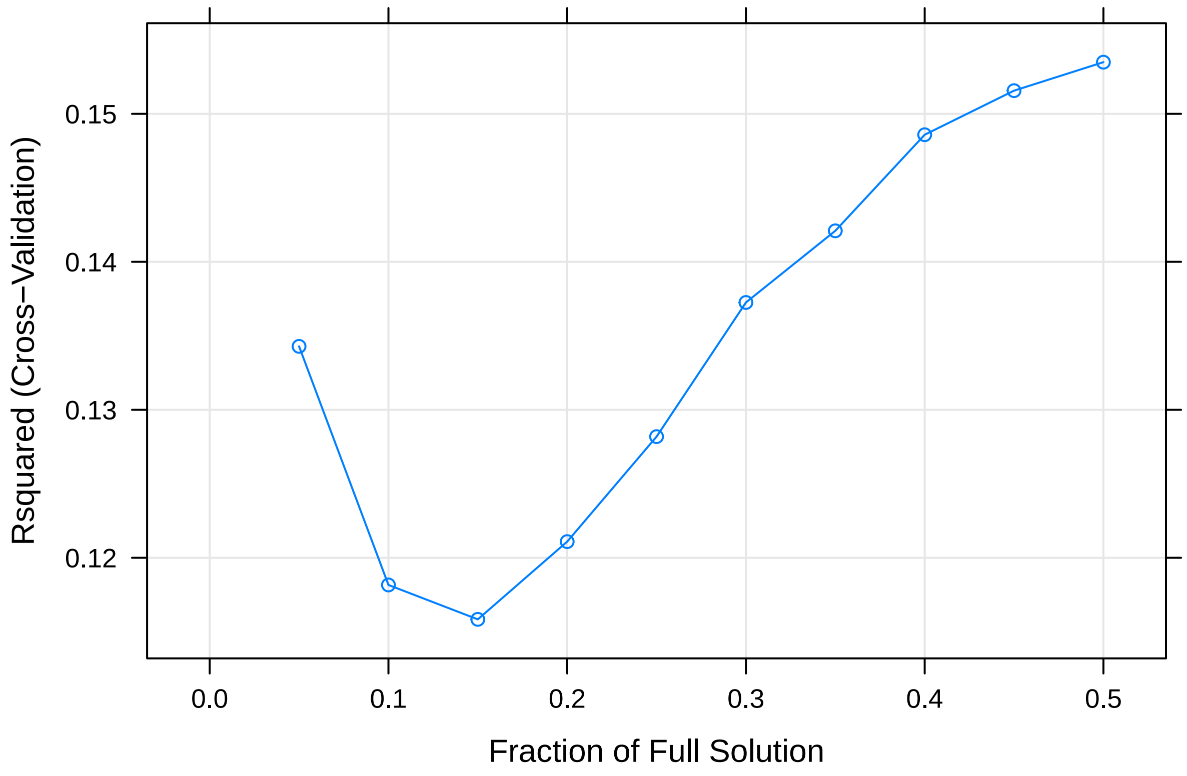
```r
lasso.fit <- train(x=X.train,
                   y=y.train,
                   method="lasso",
                   metric="Rsquared",
                   tuneGrid = data.frame(fraction=seq(0,0.5,by=0.05)),
                   trControl=trainControl(method = "cv",number=10),
                   preProcess=c("center", "scale")
                 )

lasso.fit
```

```
## The lasso
##
## 117 samples
## 110 predictors
##
## Pre-processing: centered (110), scaled (110)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 106, 105, 105, 105, 106, 106, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE          Rsquared   MAE
##   0.00      1.461272e+01        NaN  1.162243e+01
##   0.05      6.790543e+13  0.1342870  1.960261e+13
##   0.10      1.358019e+14  0.1181636  3.920262e+13
##   0.15      2.036983e+14  0.1158398  5.880263e+13
##   0.20      2.715947e+14  0.1210925  7.840264e+13
##   0.25      3.394911e+14  0.1281866  9.800265e+13
##   0.30      4.073875e+14  0.1372525  1.176027e+14
##   0.35      4.752840e+14  0.1420937  1.372027e+14
##   0.40      5.431804e+14  0.1485860  1.568027e+14
##   0.45      6.110768e+14  0.1515643  1.764027e+14
##   0.50      6.789732e+14  0.1534889  1.960027e+14
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was fraction = 0.5.
```

```r
lasso.fit$bestTune
```

```
##    fraction
## 11      0.5
```

```r
plot(lasso.fit)
```



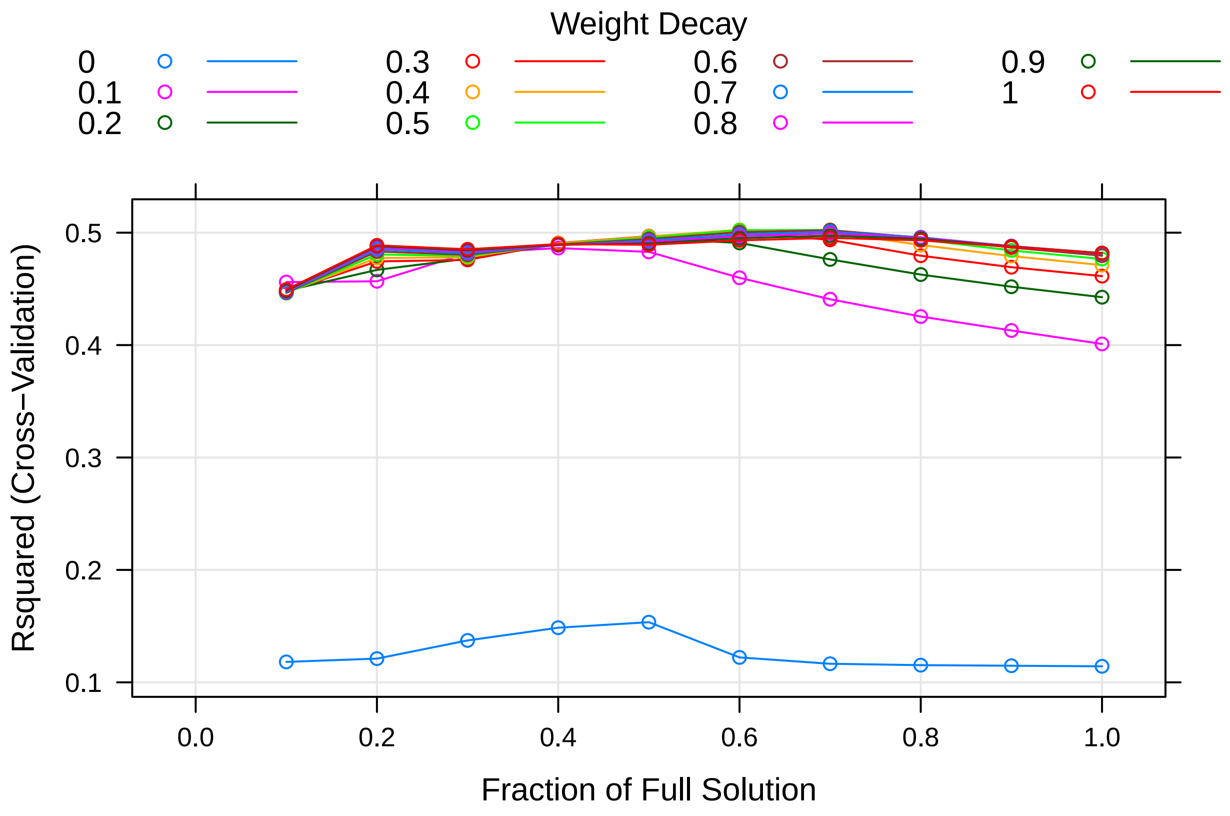
```
set.seed(786)
# tune enet model
enet.fit <- train(x=X.train,
                  y=y.train,
                  method="enet",
                  metric="Rsquared",
                  tuneGrid = expand.grid(fraction=seq(0,1,by=0.1), lambda=seq(0,1,by=0.1)),
                  trControl=trainControl(method = "cv",number=10),
                  preProcess=c("center", "scale")
                 )

enet.fit
```

```
## Elasticnet
##
## 117 samples
## 110 predictors
##
## Pre-processing: centered (110), scaled (110)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 106, 105, 105, 105, 106, 106, ...
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE          Rsquared   MAE
##   0.0     0.0       1.461272e+01        NaN  1.162243e+01
##   0.0     0.1       1.358019e+14  0.1181636  3.920262e+13
##   0.0     0.2       2.715947e+14  0.1210925  7.840264e+13
##   0.0     0.3       4.073875e+14  0.1372525  1.176027e+14
##   0.0     0.4       5.431804e+14  0.1485860  1.568027e+14
##   0.0     0.5       6.789732e+14  0.1534889  1.960027e+14
##   0.0    0.6    8.147661e+14  0.1221618  2.352027e+14
##   0.0    0.7    9.505589e+14  0.1165096  2.744027e+14
##   0.0    0.8    1.086352e+15  0.1152884  3.136027e+14
##   0.0    0.9    1.222145e+15  0.1147560  3.528028e+14
##   0.0    1.0    1.357937e+15  0.1142670  3.920028e+14
##   0.1    0.0    1.461272e+01        NaN  1.162243e+01
##   0.1    0.1    1.174780e+01  0.4560660  8.547924e+00
##   0.1    0.2    1.128228e+01  0.4567847  7.903859e+00
##   0.1    0.3    1.112004e+01  0.4817382  7.952052e+00
##   0.1    0.4    1.116090e+01  0.4862802  8.144424e+00
##   0.1    0.5    1.134600e+01  0.4828980  8.356659e+00
##   0.1    0.6    1.177955e+01  0.4598641  8.755711e+00
##   0.1    0.7    1.215731e+01  0.4407769  9.117553e+00
##   0.1    0.8    1.248846e+01  0.4253997  9.380539e+00
##   0.1    0.9    1.278433e+01  0.4129785  9.609654e+00
##   0.1    1.0    1.307559e+01  0.4010144  9.842309e+00
##   0.2    0.0    1.461272e+01        NaN  1.162243e+01
##   0.2    0.1    1.199338e+01  0.4483824  8.885793e+00
##   0.2    0.2    1.122765e+01  0.4668804  7.867286e+00
##   0.2    0.3    1.118931e+01  0.4769086  7.966145e+00
##   0.2    0.4    1.115283e+01  0.4898601  8.109737e+00
##   0.2    0.5    1.121380e+01  0.4947837  8.266779e+00
##   0.2    0.6    1.139984e+01  0.4908794  8.403479e+00
##   0.2    0.7    1.171936e+01  0.4762297  8.700399e+00
##   0.2    0.8    1.201796e+01  0.4627385  8.984932e+00
##   0.2    0.9    1.228207e+01  0.4519167  9.204006e+00
##   0.2    1.0    1.252247e+01  0.4426120  9.395175e+00
##   0.3    0.0    1.461272e+01        NaN  1.162243e+01
##   0.3    0.1    1.205633e+01  0.4476169  9.023007e+00
##   0.3    0.2    1.119102e+01  0.4744092  7.890374e+00
##   0.3    0.3    1.124332e+01  0.4757735  7.952545e+00
##   0.3    0.4    1.120933e+01  0.4907321  8.130941e+00
##   0.3    0.5    1.126181e+01  0.4968430  8.297258e+00
##   0.3    0.6    1.136627e+01  0.5003137  8.413138e+00
##   0.3    0.7    1.161553e+01  0.4936135  8.587954e+00
##   0.3    0.8    1.192758e+01  0.4795287  8.851330e+00
##   0.3    0.9    1.219473e+01  0.4692966  9.094764e+00
##   0.3    1.0    1.242482e+01  0.4613298  9.289079e+00
##   0.4    0.0    1.461272e+01        NaN  1.162243e+01
##   0.4    0.1    1.208297e+01  0.4466454  9.083181e+00
##   0.4    0.2    1.119476e+01  0.4774469  7.892279e+00
##   0.4    0.3    1.126480e+01  0.4780977  7.928803e+00
##   0.4    0.4    1.129394e+01  0.4904605  8.172674e+00
##   0.4    0.5    1.137279e+01  0.4966126  8.369748e+00
##   0.4    0.6    1.147057e+01  0.5025237  8.493729e+00
##   0.4    0.7    1.168565e+01  0.5004571  8.645898e+00
##   0.4    0.8    1.198753e+01  0.4890132  8.867928e+00
##   0.4    0.9    1.226493e+01  0.4791725  9.117997e+00
##   0.4    1.0    1.250699e+01  0.4711338  9.342353e+00
##   0.5    0.0    1.461272e+01        NaN  1.162243e+01
##   0.5    0.1    1.209136e+01  0.4462680  9.103379e+00
##   0.5    0.2    1.118549e+01  0.4806604  7.867273e+00
##   0.5    0.3    1.129283e+01  0.4792371  7.917353e+00
##   0.5    0.4    1.139194e+01  0.4898455  8.240788e+00
##   0.5     0.5       1.151586e+01  0.4955996  8.480695e+00
##   0.5     0.6       1.162994e+01  0.5020806  8.615972e+00
##   0.5     0.7       1.184713e+01  0.5024430  8.783202e+00
##   0.5     0.8       1.214099e+01  0.4938037  9.002360e+00
##   0.5     0.9       1.243341e+01  0.4842337  9.278943e+00
##   0.5     1.0       1.269048e+01  0.4765397  9.538792e+00
##   0.6     0.0       1.461272e+01        NaN  1.162243e+01
##   0.6     0.1       1.208972e+01  0.4465556  9.099953e+00
##   0.6     0.2       1.117610e+01  0.4832594  7.837067e+00
##   0.6     0.3       1.132099e+01  0.4803361  7.909903e+00
##   0.6     0.4       1.150076e+01  0.4893696  8.306793e+00
##   0.6     0.5       1.167750e+01  0.4943419  8.599148e+00
##   0.6     0.6       1.182466e+01  0.5003983  8.756518e+00
##   0.6     0.7       1.205862e+01  0.5022022  8.956576e+00
##   0.6     0.8       1.236015e+01  0.4955709  9.217107e+00
##   0.6     0.9       1.266470e+01  0.4867716  9.515938e+00
##   0.6     1.0       1.294065e+01  0.4795180  9.798881e+00
##   0.7     0.0       1.461272e+01        NaN  1.162243e+01
##   0.7     0.1       1.208224e+01  0.4469407  9.086645e+00
##   0.7     0.2       1.116929e+01  0.4852350  7.804197e+00
##   0.7     0.3       1.135107e+01  0.4816836  7.908633e+00
##   0.7     0.4       1.162213e+01  0.4889297  8.384935e+00
##   0.7     0.5       1.185328e+01  0.4930018  8.721644e+00
##   0.7     0.6       1.204433e+01  0.4985102  8.912435e+00
##   0.7     0.7       1.230412e+01  0.5009427  9.182065e+00
##   0.7     0.8       1.261815e+01  0.4959287  9.488062e+00
##   0.7     0.9       1.293996e+01  0.4878788  9.795584e+00
##   0.7     1.0       1.323834e+01  0.4810652  1.009804e+01
##   0.8     0.0       1.461272e+01        NaN  1.162243e+01
##   0.8     0.1       1.206811e+01  0.4476790  9.063588e+00
##   0.8     0.2       1.116463e+01  0.4868431  7.768901e+00
##   0.8     0.3       1.138407e+01  0.4831699  7.912801e+00
##   0.8     0.4       1.174533e+01  0.4888881  8.465786e+00
##   0.8     0.5       1.204078e+01  0.4916686  8.854128e+00
##   0.8     0.6       1.227822e+01  0.4967980  9.113438e+00
##   0.8     0.7       1.257221e+01  0.4992868  9.435123e+00
##   0.8     0.8       1.290686e+01  0.4954399  9.766742e+00
##   0.8     0.9       1.325148e+01  0.4880354  1.009341e+01
##   0.8     1.0       1.357141e+01  0.4817329  1.041612e+01
##   0.9     0.0       1.461272e+01        NaN  1.162243e+01
##   0.9     0.1       1.204940e+01  0.4483771  9.035093e+00
##   0.9     0.2       1.116177e+01  0.4880689  7.731852e+00
##   0.9     0.3       1.142396e+01  0.4843389  7.918399e+00
##   0.9     0.4       1.186864e+01  0.4892936  8.542601e+00
##   0.9     0.5       1.223885e+01  0.4903595  8.995320e+00
##   0.9     0.6       1.252730e+01  0.4950438  9.333403e+00
##   0.9     0.7       1.285825e+01  0.4973112  9.693645e+00
##   0.9     0.8       1.322112e+01  0.4944646  1.005931e+01
##   0.9     0.9       1.359040e+01  0.4877176  1.040854e+01
##   0.9     1.0       1.393148e+01  0.4818460  1.074867e+01
##   1.0     0.0       1.461272e+01        NaN  1.162243e+01
##   1.0     0.1       1.202690e+01  0.4493657  9.004348e+00
##   1.0     0.2       1.116484e+01  0.4887120  7.708108e+00
##   1.0     0.3       1.147188e+01  0.4852200  7.927144e+00
##   1.0      0.4       1.199165e+01  0.4898434  8.619211e+00
##   1.0      0.5       1.244455e+01  0.4892267  9.161889e+00
##   1.0      0.6       1.278751e+01  0.4931700  9.565456e+00
##   1.0      0.7       1.315725e+01  0.4953460  9.964516e+00
##   1.0      0.8       1.355341e+01  0.4931895  1.035892e+01
##   1.0      0.9       1.394499e+01  0.4873878  1.072357e+01
##   1.0      1.0       1.431243e+01  0.4816042  1.107811e+01
##
## Rsquared was used to select the optimal model using the largest value.
## The final values used for the model were fraction = 0.6 and lambda = 0.4.
```

```
enet.fit$bestTune
```

```
##    fraction lambda
## 51      0.6    0.4
```

```
plot(enet.fit)
```



Comparing the summary below we could see the best model as elastic net against the training data as the R^2 is 0.503.

```
set.seed(786)
summary(resamples(list(PLS=pls.fit, Ridge=ridge.fit, Lasso=lasso.fit, Elasticnet=enet.fit)))
```

```
##
## Call:
## summary.resamples(object = resamples(list(PLS = pls.fit, Ridge =
##  ridge.fit, Lasso = lasso.fit, Elasticnet = enet.fit)))
##
## Models: PLS, Ridge, Lasso, Elasticnet
## Number of resamples: 10
```

```
##
## MAE
##                 Min.   1st Qu.      Median         Mean      3rd Qu.
## PLS         4.394289  6.610297    8.479611 8.071552e+00     9.875764
## Ridge       6.210415  8.356001    9.765291 1.074867e+01    11.134924
## Lasso      25.491473 90.877616 2249.801725 1.960027e+14 77661.980610
## Elasticnet  5.054485  6.645612    8.115724 8.493729e+00     8.869695
##                    Max. NA's
## PLS        1.075878e+01    0
## Ridge      1.920052e+01    0
## Lasso      1.960027e+15    0
## Elasticnet 1.586292e+01    0
##
## RMSE
##                 Min.    1st Qu.     Median         Mean      3rd Qu.
## PLS         6.290607   8.377139   12.19528 1.119413e+01     13.43221
## Ridge       8.337455  10.191079   13.74461 1.393148e+01     16.99641
## Lasso      44.734039 170.204105 4149.21338 6.789732e+14 171049.39095
## Elasticnet  7.363015   9.416748   11.54396 1.147057e+01     11.90036
##                    Max. NA's
## PLS        1.594366e+01    0
## Ridge      2.185108e+01    0
## Lasso      6.789732e+15    0
## Elasticnet 1.842877e+01    0
##
## Rsquared
##                  Min.    1st Qu.     Median      Mean   3rd Qu.      Max. NA's
## PLS        0.16129728 0.25233183 0.56351166 0.4761420 0.6036028 0.8025506    0
## Ridge      0.08380318 0.28691652 0.51368868 0.4818460 0.6857065 0.8412046    0
## Lasso      0.03511831 0.04684349 0.08472472 0.1534889 0.2044714 0.5015686    0
## Elasticnet 0.09920794 0.33110233 0.54300031 0.5025237 0.6672628 0.7914440    0
```

Next is to compare the accuracies for all the model against the test data.

```
accuracy <- function(models, predictor, response) {
  ac <- list()
  i <- 1
  for (m in models) {
    prediction <- predict(m, newdata = predictor)
    ac[[i]] <- postResample(pred=prediction, obs = response)
    i <- i+1
  }
  names(ac) <- c("pls","ridge","lasso","enet")
  return(ac)
}

models <- list(pls.fit, ridge.fit, lasso.fit, enet.fit)
accuracy(models, X.test, y.test)
```

```
## $pls
##       RMSE   Rsquared        MAE
## 12.4644848  0.4761679  9.4899066
##
## $ridge
##       RMSE   Rsquared        MAE
## 13.1757984  0.5906079  9.7296879
##
## $lasso
##         RMSE     Rsquared          MAE
## 3.919983e+03 4.075392e-02 1.181761e+03
##
## $enet
##       RMSE   Rsquared        MAE
## 11.6267322  0.5639022  8.7051313
```

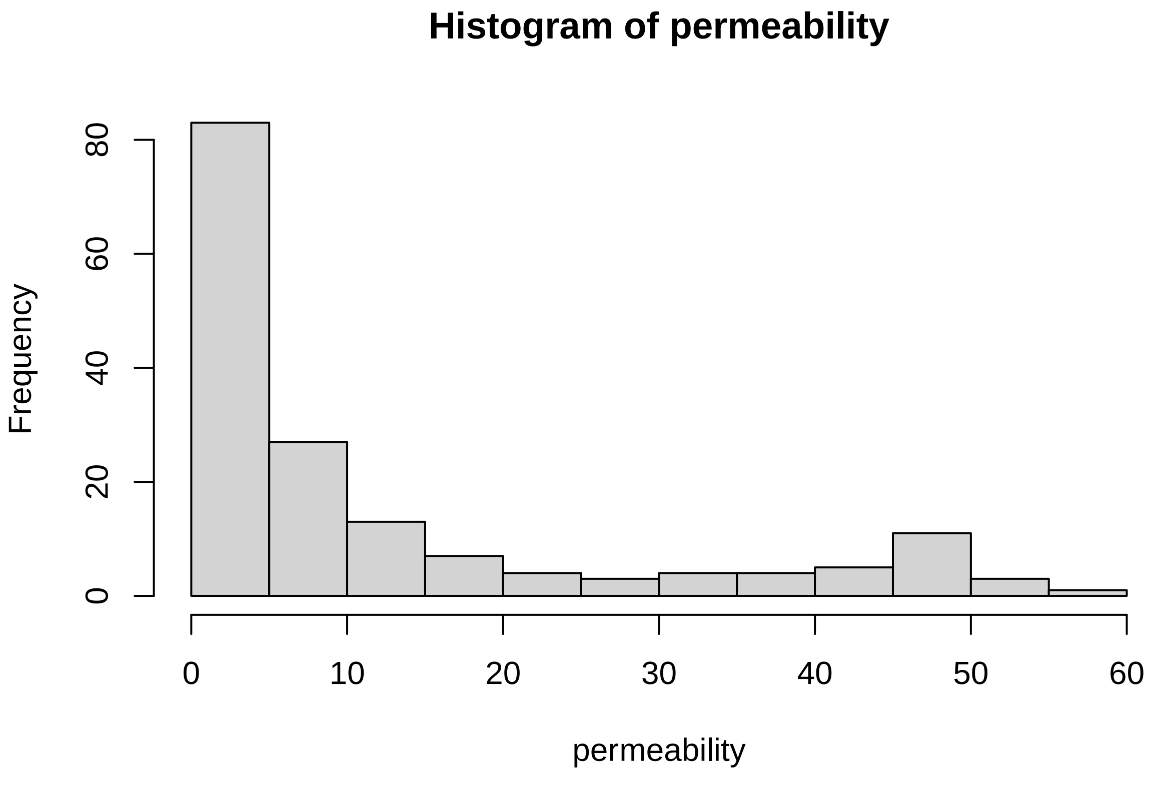
The test set evaluation seems to show Ridge model is best with R^2=0.59 but the cross validation earlier showed the best model as elastic net. I would prefer to go with cross validation as it is closer to true distribution. With this explanation, we can conclude that elastic net predicted the test set with best accuracy having RMSE = 11.62 and R^2 = 0.56

(f)

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

I would not recommend any of the models described above to replace the permeability laboratory experiment. The MAE of all the models are roughly between 8 and 10 that means model predictions of on avg 8 to 10 off. If we look at the permiability histogram most of the values are under 10 so models accuracies are not good enough to replace lab test.

```
hist(permeability)
```



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, the 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:

(a)

Start R and use these commands to load the data:

```
data(ChemicalManufacturingProcess)
```

```
glimpse(ChemicalManufacturingProcess)
```

```
## Rows: 176
## Columns: 58
## $ Yield                  <dbl> 38.00, 42.44, 42.03, 41.42, 42.49, 43.57, 43.12~
## $ BiologicalMaterial01   <dbl> 6.25, 8.01, 8.01, 8.01, 7.47, 6.12, 7.48, 6.94,~
## $ BiologicalMaterial02   <dbl> 49.58, 60.97, 60.97, 60.97, 63.33, 58.36, 64.47~
## $ BiologicalMaterial03   <dbl> 56.97, 67.48, 67.48, 67.48, 72.25, 65.31, 72.41~
## $ BiologicalMaterial04   <dbl> 12.74, 14.65, 14.65, 14.65, 14.02, 15.17, 13.82~
## $ BiologicalMaterial05   <dbl> 19.51, 19.36, 19.36, 19.36, 17.91, 21.79, 17.71~
## $ BiologicalMaterial06   <dbl> 43.73, 53.14, 53.14, 53.14, 54.66, 51.23, 54.45~
## $ BiologicalMaterial07   <dbl> 100, 100, 100, 100, 100, 100, 100, 100, 100, 10~
## $ BiologicalMaterial08   <dbl> 16.66, 19.04, 19.04, 19.04, 18.22, 18.30, 18.72~
## $ BiologicalMaterial09   <dbl> 11.44, 12.55, 12.55, 12.55, 12.80, 12.13, 12.95~
## $ BiologicalMaterial10   <dbl> 3.46, 3.46, 3.46, 3.46, 3.05, 3.78, 3.04, 3.85,~
## $ BiologicalMaterial11   <dbl> 138.09, 153.67, 153.67, 153.67, 147.61, 151.88,~
## $ BiologicalMaterial12   <dbl> 18.83, 21.05, 21.05, 21.05, 21.05, 20.76, 20.75~
## $ ManufacturingProcess01 <dbl> NA, 0.0, 0.0, 0.0, 10.7, 12.0, 11.5, 12.0, 12.0~
## $ ManufacturingProcess02 <dbl> NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ ManufacturingProcess03 <dbl> NA, NA, NA, NA, NA, NA, 1.56, 1.55, 1.56, 1.55,~
## $ ManufacturingProcess04 <dbl> NA, 917, 912, 911, 918, 924, 933, 929, 928, 938~
## $ ManufacturingProcess05 <dbl> NA, 1032.2, 1003.6, 1014.6, 1027.5, 1016.8, 988~
## $ ManufacturingProcess06 <dbl> NA, 210.0, 207.1, 213.3, 205.7, 208.9, 210.0, 2~
## $ ManufacturingProcess07 <dbl> NA, 177, 178, 177, 178, 178, 177, 178, 177, 177~
## $ ManufacturingProcess08 <dbl> NA, 178, 178, 177, 178, 178, 178, 178, 177, 177~
## $ ManufacturingProcess09 <dbl> 43.00, 46.57, 45.07, 44.92, 44.96, 45.32, 49.36~
## $ ManufacturingProcess10 <dbl> NA, NA, NA, NA, NA, NA, 11.6, 10.2, 9.7, 10.1, ~
## $ ManufacturingProcess11 <dbl> NA, NA, NA, NA, NA, NA, 11.5, 11.3, 11.1, 10.2,~
## $ ManufacturingProcess12 <dbl> NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ ManufacturingProcess13 <dbl> 35.5, 34.0, 34.8, 34.8, 34.6, 34.0, 32.4, 33.6,~
## $ ManufacturingProcess14 <dbl> 4898, 4869, 4878, 4897, 4992, 4985, 4745, 4854,~
## $ ManufacturingProcess15 <dbl> 6108, 6095, 6087, 6102, 6233, 6222, 5999, 6105,~
## $ ManufacturingProcess16 <dbl> 4682, 4617, 4617, 4635, 4733, 4786, 4486, 4626,~
## $ ManufacturingProcess17 <dbl> 35.5, 34.0, 34.8, 34.8, 33.9, 33.4, 33.8, 33.6,~
## $ ManufacturingProcess18 <dbl> 4865, 4867, 4877, 4872, 4886, 4862, 4758, 4766,~
## $ ManufacturingProcess19 <dbl> 6049, 6097, 6078, 6073, 6102, 6115, 6013, 6022,~
## $ ManufacturingProcess20 <dbl> 4665, 4621, 4621, 4611, 4659, 4696, 4522, 4552,~
## $ ManufacturingProcess21 <dbl> 0.0, 0.0, 0.0, 0.0, -0.7, -0.6, 1.4, 0.0, 0.0, ~
## $ ManufacturingProcess22 <dbl> NA, 3, 4, 5, 8, 9, 1, 2, 3, 4, 6, 7, 8, 10, 11,~
## $ ManufacturingProcess23 <dbl> NA, 0, 1, 2, 4, 1, 1, 2, 3, 1, 3, 4, 1, 2, 3, 4~
## $ ManufacturingProcess24 <dbl> NA, 3, 4, 5, 18, 1, 1, 2, 3, 4, 6, 7, 8, 2, 15,~
## $ ManufacturingProcess25 <dbl> 4873, 4869, 4897, 4892, 4930, 4871, 4795, 4806,~
## $ ManufacturingProcess26 <dbl> 6074, 6107, 6116, 6111, 6151, 6128, 6057, 6059,~
## $ ManufacturingProcess27 <dbl> 4685, 4630, 4637, 4630, 4684, 4687, 4572, 4586,~
## $ ManufacturingProcess28 <dbl> 10.7, 11.2, 11.1, 11.1, 11.3, 11.4, 11.2, 11.1,~
## $ ManufacturingProcess29 <dbl> 21.0, 21.4, 21.3, 21.3, 21.6, 21.7, 21.2, 21.2,~
## $ ManufacturingProcess30 <dbl> 9.9, 9.9, 9.4, 9.4, 9.0, 10.1, 11.2, 10.9, 10.5~
## $ ManufacturingProcess31 <dbl> 69.1, 68.7, 69.3, 69.3, 69.4, 68.2, 67.6, 67.9,~
## $ ManufacturingProcess32 <dbl> 156, 169, 173, 171, 171, 173, 159, 161, 160, 16~
## $ ManufacturingProcess33 <dbl> 66, 66, 66, 68, 70, 70, 65, 65, 65, 66, 67, 67,~
## $ ManufacturingProcess34 <dbl> 2.4, 2.6, 2.6, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.~
## $ ManufacturingProcess35 <dbl> 486, 508, 509, 496, 468, 490, 475, 478, 491, 48~
## $ ManufacturingProcess36 <dbl> 0.019, 0.019, 0.018, 0.018, 0.017, 0.018, 0.019~
## $ ManufacturingProcess37 <dbl> 0.5, 2.0, 0.7, 1.2, 0.2, 0.4, 0.8, 1.0, 1.2, 1.~
## $ ManufacturingProcess38 <dbl> 3, 2, 2, 2, 2, 2, 2, 2, 3, 3, 2, 3, 3, 3, 3, 3,~
## $ ManufacturingProcess39 <dbl> 7.2, 7.2, 7.2, 7.2, 7.3, 7.2, 7.3, 7.3, 7.4, 7.~
## $ ManufacturingProcess40 <dbl> NA, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0~
## $ ManufacturingProcess41 <dbl> NA, 0.15, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0~
## $ ManufacturingProcess42 <dbl> 11.6, 11.1, 12.0, 10.6, 11.0, 11.5, 11.7, 11.4,~
## $ ManufacturingProcess43 <dbl> 3.0, 0.9, 1.0, 1.1, 1.1, 2.2, 0.7, 0.8, 0.9, 0.~
## $ ManufacturingProcess44 <dbl> 1.8, 1.9, 1.8, 1.8, 1.7, 1.8, 2.0, 2.0, 1.9, 1.~
## $ ManufacturingProcess45 <dbl> 2.4, 2.2, 2.3, 2.1, 2.1, 2.0, 2.2, 2.2, 2.1, 2.~
```

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.

(b)

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).

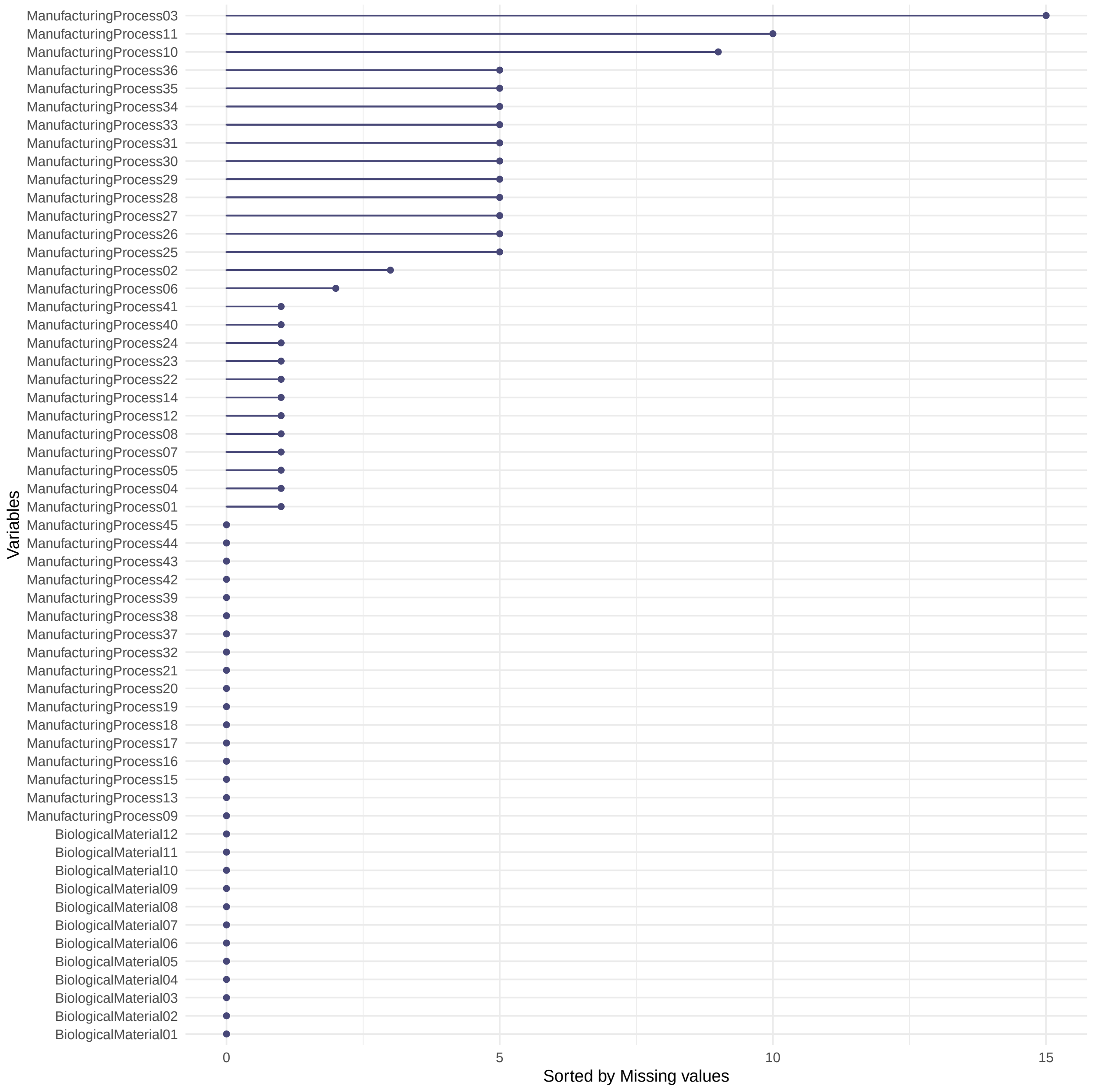
We will first see all the variables having any of the missing values. We have used below complete.cases() function to find the the missing values.

```
# columns having missing values
colnames(ChemicalManufacturingProcess)[!complete.cases(t(ChemicalManufacturingProcess))]
```

```
##  [1] "ManufacturingProcess01" "ManufacturingProcess02" "ManufacturingProcess03"
##  [4] "ManufacturingProcess04" "ManufacturingProcess05" "ManufacturingProcess06"
##  [7] "ManufacturingProcess07" "ManufacturingProcess08" "ManufacturingProcess10"
## [10] "ManufacturingProcess11" "ManufacturingProcess12" "ManufacturingProcess14"
## [13] "ManufacturingProcess22" "ManufacturingProcess23" "ManufacturingProcess24"
## [16] "ManufacturingProcess25" "ManufacturingProcess26" "ManufacturingProcess27"
## [19] "ManufacturingProcess28" "ManufacturingProcess29" "ManufacturingProcess30"
## [22] "ManufacturingProcess31" "ManufacturingProcess33" "ManufacturingProcess34"
## [25] "ManufacturingProcess35" "ManufacturingProcess36" "ManufacturingProcess40"
## [28] "ManufacturingProcess41"
```

So there are 28 columns having missing values. Here is the plot for missing values of all the predictors.

```
gg_miss_var(ChemicalManufacturingProcess[,-c(1)]) + labs(y = "Sorted by Missing values")
```



We will next use preProcess() method to impute the missing values using knnImpute (K nearest neighbor).

```
pre.proc <- preProcess(ChemicalManufacturingProcess[,c(-1)], method = "knnImpute")
chem_df <- predict(pre.proc, ChemicalManufacturingProcess[,c(-1)])
```

```
# columns having missing values
colnames(chem_df)[!complete.cases(t(chem_df))]
```

```
## character(0)
```

(c)

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?

We will first filter out the predictors that have low frequencies using the `nearZeroVar` function from the caret package. After applying this function we see 1 column is removed and 56 predictors are left for modeling.

```
chem.remove.pred <- nearZeroVar(chem_df)
chem_df <- chem_df[,-chem.remove.pred]
length(chem.remove.pred) %>% paste('columns are removed. ', dim(chem_df)[2], ' predictors are left for 
```

```
## [1] "1 columns are removed.  56  predictors are left for modeling."
```

We will now look into pairwise correlation above 0.90 and remove the predictors having correlation with cutoff 0.90.

```
chem.corr.90 <- findCorrelation(cor(chem_df), cutoff=0.90)
chem_df <- chem_df[,-chem.corr.90]
length(chem.corr.90) %>% paste('columns having correlation 0.90 or more are removed. ', dim(chem_df)[2]
```

```
## [1] "10 columns having correlation 0.90 or more are removed.  46  predictors are left for modeling."
```

Next step is to split the data in training and testing set. We reserve 70% for training and 30% for testing. After split we will fit elastic net model.

```
set.seed(786)

pre.proc <- preProcess(chem_df, method = c("center", "scale"))
chem_df <- predict(pre.proc, chem_df)

# partition
chem.part <- createDataPartition(ChemicalManufacturingProcess$Yield, p=0.70, list = FALSE)

# predictor
X.train <- chem_df[chem.part,]
X.test <- chem_df[-chem.part,]

# response
y.train <- ChemicalManufacturingProcess$Yield[chem.part]
y.test <- ChemicalManufacturingProcess$Yield[-chem.part]

# tune elastic net model
chem.enet.fit <- train(x=X.train,
                       y=y.train,
                       method="glmnet",
                       metric="Rsquared",
                       trControl=trainControl(method = "cv",number=10),
                       tuneLength = 5
                 )


chem.enet.fit
```

```
## glmnet
##
## 124 samples
##  46 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 112, 112, 111, 112, 112, 112, ...
## Resampling results across tuning parameters:
```

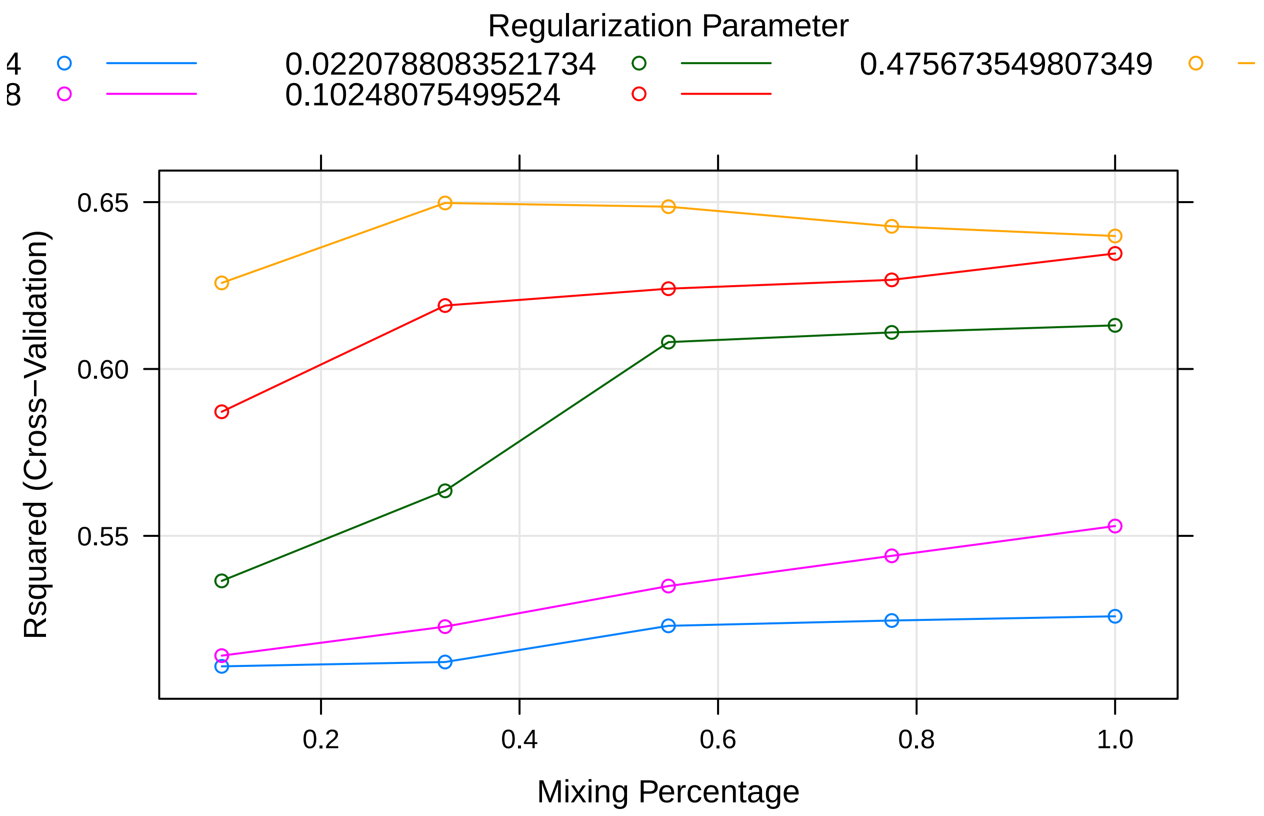
```
##
##   alpha  lambda       RMSE      Rsquared   MAE
##   0.100  0.001024807  1.652735  0.5109039  1.2180963
##   0.100  0.004756735  1.787829  0.5141078  1.2448950
##   0.100  0.022078808  1.928153  0.5365144  1.2480137
##   0.100  0.102480755  1.713449  0.5871842  1.1313255
##   0.100  0.475673550  1.462774  0.6257977  1.0229502
##   0.325  0.001024807  1.674895  0.5121971  1.2205224
##   0.325  0.004756735  1.736684  0.5227997  1.2158273
##   0.325  0.022078808  1.645135  0.5634960  1.1318892
##   0.325  0.102480755  1.597280  0.6190164  1.0582109
##   0.325  0.475673550  1.146042  0.6497417  0.9475750
##   0.550  0.001024807  1.618105  0.5230439  1.1957915
##   0.550  0.004756735  1.627257  0.5349631  1.1697813
##   0.550  0.022078808  1.437392  0.6080512  1.0404923
##   0.550  0.102480755  1.499209  0.6240617  1.0206172
##   0.550  0.475673550  1.165737  0.6486340  0.9751643
##   0.775  0.001024807  1.619876  0.5246330  1.1934018
##   0.775  0.004756735  1.577763  0.5440200  1.1437141
##   0.775  0.022078808  1.466964  0.6109600  1.0399092
##   0.775  0.102480755  1.393329  0.6267072  0.9884416
##   0.775  0.475673550  1.225313  0.6427517  1.0185178
##   1.000  0.001024807  1.623632  0.5259057  1.1914845
##   1.000  0.004756735  1.549576  0.5529405  1.1249330
##   1.000  0.022078808  1.553233  0.6130731  1.0571319
##   1.000  0.102480755  1.225924  0.6346165  0.9461467
##   1.000  0.475673550  1.290412  0.6398509  1.0723391
##
## Rsquared was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.325 and lambda = 0.4756735.
```

```
chem.enet.fit$bestTune
```

```
##    alpha    lambda
## 10 0.325 0.4756735
```

From the elastic net model we see the best alpha and lambda came up as 0.32 AND 0.48 respectively. We have used R^2 to select the optimal model.

```
plot(chem.enet.fit)
```



Now we will find the RSquare and RMSE values for besttune model. It comes out R^2 as 63% and RMSE as 1.15. We see the model is able to explain 63% of variance of the data.

```
set.seed(786)
data.frame(RSquared=chem.enet.fit[["results"]][["Rsquared"]][as.numeric(rownames(chem.enet.fit$bestTune
           RMSE=chem.enet.fit[["results"]][["RMSE"]][as.numeric(rownames(chem.enet.fit$bestTune))])
```

```
##    RSquared     RMSE
## 1 0.6497417 1.146042
```

(d)

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?

Lets do the test prediction and check the values of R^2, RMSE and MAE parameters.

```
set.seed(786)
enet.pred <- predict(chem.enet.fit, newdata = X.test)
(pred.result <- postResample(pred = enet.pred, obs=y.test))
```

```
##      RMSE  Rsquared       MAE
## 1.3301919 0.5123376 1.0823407
```

(e)

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

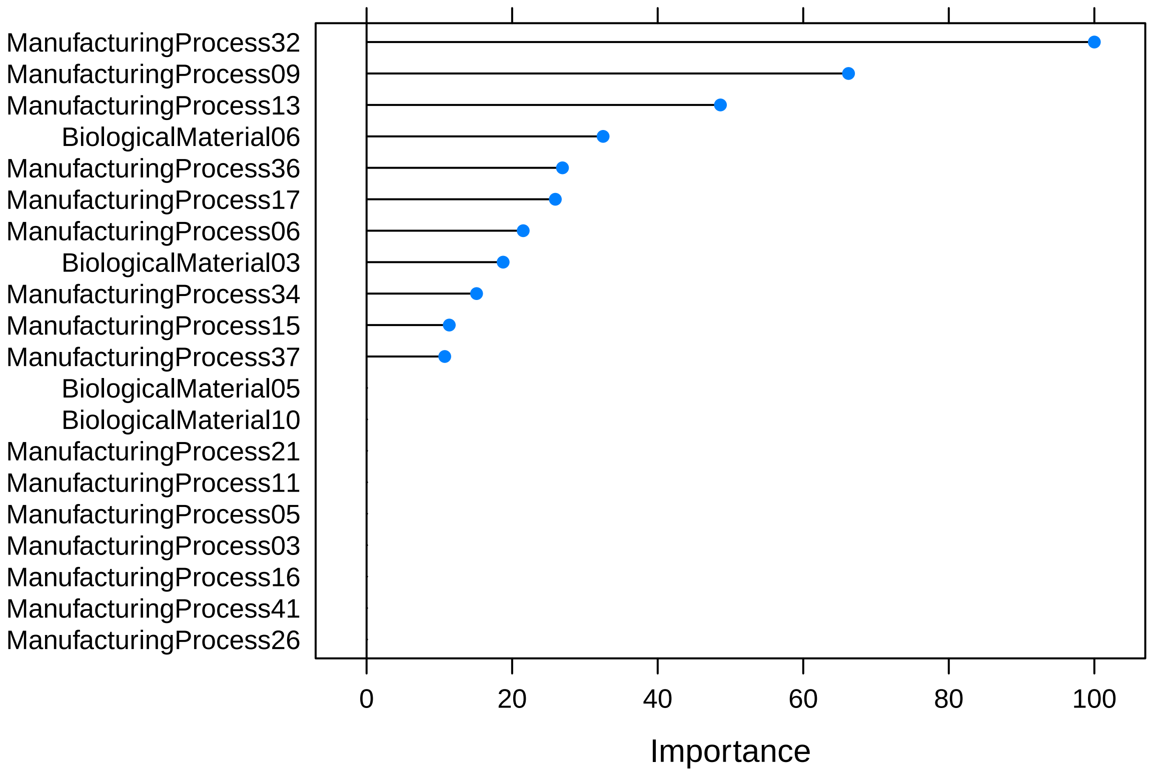
`caret:varImp` calculation of variable importance for regression and classification models, is generic method for calculating variable importance for objects produced by train and method specific methods. The number returned is a relative measure of variable importance.

We can see here that the most contribution predictor is ManufacturingProcess32 and we can conclude that Manufacturing process predictors dominate the list.

```
varImp(chem.enet.fit)
```

```
## glmnet variable importance
##
##   only 20 most important variables shown (out of 46)
##
##                        Overall
## ManufacturingProcess32  100.00
## ManufacturingProcess09   66.22
## ManufacturingProcess13   48.62
## BiologicalMaterial06     32.50
## ManufacturingProcess36   26.92
## ManufacturingProcess17   25.93
## ManufacturingProcess06   21.52
## BiologicalMaterial03     18.76
## ManufacturingProcess34   15.12
## ManufacturingProcess15   11.36
## ManufacturingProcess37   10.74
## ManufacturingProcess10    0.00
## ManufacturingProcess07    0.00
## ManufacturingProcess33    0.00
## ManufacturingProcess02    0.00
## ManufacturingProcess16    0.00
## ManufacturingProcess05    0.00
## BiologicalMaterial05      0.00
## ManufacturingProcess30    0.00
## ManufacturingProcess08    0.00
```

```
plot(varImp(chem.enet.fit), top=20)
```



(f)

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?

A positive coefficient shows that is corresponding predictore increases, response mean will also increase while the negative coefficient shows vice versa. Among the positive coefficients, `ManufacturingProcess32` has the highest coefficient value.

```
coeffs = coef(chem.enet.fit$finalModel, chem.enet.fit$bestTune$lambda)
(coeffs.df = data.frame(cbind(variables = coeffs@Dimnames[[1]][coeffs@i+1], coef = coeffs@x)))
```

```
##                 variables                coef
## 1             (Intercept)    40.1826173671972
## 2    BiologicalMaterial03   0.096166348616399
## 3    BiologicalMaterial06   0.166570232040782
## 4  ManufacturingProcess06   0.110297740389495
## 5  ManufacturingProcess09   0.339419229880968
## 6  ManufacturingProcess13  -0.249207854549869
## 7  ManufacturingProcess15  0.0582484042571225
## 8  ManufacturingProcess17  -0.132925194155295
## 9  ManufacturingProcess32   0.512533203246302
## 10 ManufacturingProcess34  0.0774694027998954
## 11 ManufacturingProcess36  -0.137952195417695
## 12 ManufacturingProcess37 -0.0550415346074052
```

```
coeffs.df[coeffs.df$coef>0,]
```

```
##                 variables               coef
## 1             (Intercept)   40.1826173671972
## 2    BiologicalMaterial03  0.096166348616399
## 3    BiologicalMaterial06  0.166570232040782
## 4  ManufacturingProcess06  0.110297740389495
## 5  ManufacturingProcess09  0.339419229880968
## 7  ManufacturingProcess15 0.0582484042571225
## 9  ManufacturingProcess32  0.512533203246302
## 10 ManufacturingProcess34 0.0774694027998954
```

Among the positive coefficients, `ManufacturingProcess13` has the lowest coefficient value then the other values.

```
coeffs.df[coeffs.df$coef<0,]
```

```
##                 variables                coef
## 6  ManufacturingProcess13  -0.249207854549869
## 8  ManufacturingProcess17  -0.132925194155295
## 11 ManufacturingProcess36  -0.137952195417695
## 12 ManufacturingProcess37 -0.0550415346074052
```

Finally here is the correlation value for `ManufacturingProcess32` and `ManufacturingProcess13` with the response variable `Yield`.

```
cor(ChemicalManufacturingProcess$Yield, ChemicalManufacturingProcess$ManufacturingProcess32)
```

```
## [1] 0.6083321
```

```
cor(ChemicalManufacturingProcess$Yield, ChemicalManufacturingProcess$ManufacturingProcess13)
```

```
## [1] -0.5036797
```