Homework Set 2 in R

DATA 624-01 Group 3

Z. Palmore, K. Popkin, K. Potter, C. Nan, J. Ramalingam

7/8/2021

Table of Contents

[KJ 6.3 2](#_Toc77097730)

[Part A 2](#_Toc77097731)

[Part B 5](#_Toc77097732)

[Part C 5](#_Toc77097733)

[Part D 8](#_Toc77097734)

[Part E 11](#_Toc77097735)

[Part F 11](#_Toc77097736)

[KJ 7.2 11](#_Toc77097737)

[Part A 12](#_Toc77097738)

[KJ 7.5 18](#_Toc77097739)

[Part A 18](#_Toc77097740)

[Part B 27](#_Toc77097741)

[Part C 27](#_Toc77097742)

[KJ 8.1 27](#_Toc77097743)

[Part A 31](#_Toc77097744)

[Part B 31](#_Toc77097745)

[Part C 32](#_Toc77097746)

[Part D 33](#_Toc77097747)

[KJ 8.2 34](#_Toc77097748)

[Part A 34](#_Toc77097749)

[KJ 8.3 35](#_Toc77097750)

[Part A 35](#_Toc77097751)

[Part B 36](#_Toc77097752)

[KJ 8.7 37](#_Toc77097753)

[Part A 37](#_Toc77097754)

[Part B 45](#_Toc77097755)

[Part C 46](#_Toc77097756)

[Market Basket Analysis 46](#_Toc77097757)

[Part A 46](#_Toc77097758)

## KJ 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:

### Part A

#### Question

Start R and use these commands to load the data: > library(AppliedPredictiveModeling) > data(chemicalManufacturingProcess)

#### Code

library(AppliedPredictiveModeling)  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
head(df)

## Yield BiologicalMaterial01 BiologicalMaterial02 BiologicalMaterial03  
## 1 38.00 6.25 49.58 56.97  
## 2 42.44 8.01 60.97 67.48  
## 3 42.03 8.01 60.97 67.48  
## 4 41.42 8.01 60.97 67.48  
## 5 42.49 7.47 63.33 72.25  
## 6 43.57 6.12 58.36 65.31  
## BiologicalMaterial04 BiologicalMaterial05 BiologicalMaterial06  
## 1 12.74 19.51 43.73  
## 2 14.65 19.36 53.14  
## 3 14.65 19.36 53.14  
## 4 14.65 19.36 53.14  
## 5 14.02 17.91 54.66  
## 6 15.17 21.79 51.23  
## BiologicalMaterial07 BiologicalMaterial08 BiologicalMaterial09  
## 1 100 16.66 11.44  
## 2 100 19.04 12.55  
## 3 100 19.04 12.55  
## 4 100 19.04 12.55  
## 5 100 18.22 12.80  
## 6 100 18.30 12.13  
## BiologicalMaterial10 BiologicalMaterial11 BiologicalMaterial12  
## 1 3.46 138.09 18.83  
## 2 3.46 153.67 21.05  
## 3 3.46 153.67 21.05  
## 4 3.46 153.67 21.05  
## 5 3.05 147.61 21.05  
## 6 3.78 151.88 20.76  
## ManufacturingProcess01 ManufacturingProcess02 ManufacturingProcess03  
## 1 NA NA NA  
## 2 0.0 0 NA  
## 3 0.0 0 NA  
## 4 0.0 0 NA  
## 5 10.7 0 NA  
## 6 12.0 0 NA  
## ManufacturingProcess04 ManufacturingProcess05 ManufacturingProcess06  
## 1 NA NA NA  
## 2 917 1032.2 210.0  
## 3 912 1003.6 207.1  
## 4 911 1014.6 213.3  
## 5 918 1027.5 205.7  
## 6 924 1016.8 208.9  
## ManufacturingProcess07 ManufacturingProcess08 ManufacturingProcess09  
## 1 NA NA 43.00  
## 2 177 178 46.57  
## 3 178 178 45.07  
## 4 177 177 44.92  
## 5 178 178 44.96  
## 6 178 178 45.32  
## ManufacturingProcess10 ManufacturingProcess11 ManufacturingProcess12  
## 1 NA NA NA  
## 2 NA NA 0  
## 3 NA NA 0  
## 4 NA NA 0  
## 5 NA NA 0  
## 6 NA NA 0  
## ManufacturingProcess13 ManufacturingProcess14 ManufacturingProcess15  
## 1 35.5 4898 6108  
## 2 34.0 4869 6095  
## 3 34.8 4878 6087  
## 4 34.8 4897 6102  
## 5 34.6 4992 6233  
## 6 34.0 4985 6222  
## ManufacturingProcess16 ManufacturingProcess17 ManufacturingProcess18  
## 1 4682 35.5 4865  
## 2 4617 34.0 4867  
## 3 4617 34.8 4877  
## 4 4635 34.8 4872  
## 5 4733 33.9 4886  
## 6 4786 33.4 4862  
## ManufacturingProcess19 ManufacturingProcess20 ManufacturingProcess21  
## 1 6049 4665 0.0  
## 2 6097 4621 0.0  
## 3 6078 4621 0.0  
## 4 6073 4611 0.0  
## 5 6102 4659 -0.7  
## 6 6115 4696 -0.6  
## ManufacturingProcess22 ManufacturingProcess23 ManufacturingProcess24  
## 1 NA NA NA  
## 2 3 0 3  
## 3 4 1 4  
## 4 5 2 5  
## 5 8 4 18  
## 6 9 1 1  
## ManufacturingProcess25 ManufacturingProcess26 ManufacturingProcess27  
## 1 4873 6074 4685  
## 2 4869 6107 4630  
## 3 4897 6116 4637  
## 4 4892 6111 4630  
## 5 4930 6151 4684  
## 6 4871 6128 4687  
## ManufacturingProcess28 ManufacturingProcess29 ManufacturingProcess30  
## 1 10.7 21.0 9.9  
## 2 11.2 21.4 9.9  
## 3 11.1 21.3 9.4  
## 4 11.1 21.3 9.4  
## 5 11.3 21.6 9.0  
## 6 11.4 21.7 10.1  
## ManufacturingProcess31 ManufacturingProcess32 ManufacturingProcess33  
## 1 69.1 156 66  
## 2 68.7 169 66  
## 3 69.3 173 66  
## 4 69.3 171 68  
## 5 69.4 171 70  
## 6 68.2 173 70  
## ManufacturingProcess34 ManufacturingProcess35 ManufacturingProcess36  
## 1 2.4 486 0.019  
## 2 2.6 508 0.019  
## 3 2.6 509 0.018  
## 4 2.5 496 0.018  
## 5 2.5 468 0.017  
## 6 2.5 490 0.018  
## ManufacturingProcess37 ManufacturingProcess38 ManufacturingProcess39  
## 1 0.5 3 7.2  
## 2 2.0 2 7.2  
## 3 0.7 2 7.2  
## 4 1.2 2 7.2  
## 5 0.2 2 7.3  
## 6 0.4 2 7.2  
## ManufacturingProcess40 ManufacturingProcess41 ManufacturingProcess42  
## 1 NA NA 11.6  
## 2 0.1 0.15 11.1  
## 3 0.0 0.00 12.0  
## 4 0.0 0.00 10.6  
## 5 0.0 0.00 11.0  
## 6 0.0 0.00 11.5  
## ManufacturingProcess43 ManufacturingProcess44 ManufacturingProcess45  
## 1 3.0 1.8 2.4  
## 2 0.9 1.9 2.2  
## 3 1.0 1.8 2.3  
## 4 1.1 1.8 2.1  
## 5 1.1 1.7 2.1  
## 6 2.2 1.8 2.0

#### Response

Our code displays a table with the first 5 observations of each predictor variable. The table contains 57 predictors with 12 of those describing the input biological material and 45 describing the process predictors from 176 manufacturing runs. The response variable ‘Yield’ contains the percent yield for each run. There are missing values for several variables that will need imputation.

### Part B

#### Question

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).

#### Code

#Fill in missing values with the median of each feature  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}

#### Response

Roughly 1.04% of the data was missing. It was first checked for patterns in the missing values. There were none. The missing data points were noted as random occurrences where no special treatment was required during imputation. We then ran two methods of imputation for referencing, one by predictive mean matching and the other by median. We decided to use the data set imputed by the median of each variable. No missing values remained.

### Part C

#### Question

Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter.

#### Code

library(reshape2)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

cormat <- round(cor(df),2)  
melted\_cormat = melt(cormat)  
melted\_cormat\_df = data.frame(melted\_cormat)  
#Filter to only Yield and sort in descending order  
yield\_corr <- melted\_cormat\_df %>%   
 filter(Var2 == "Yield")  
yield\_corr$absvalue = abs(yield\_corr$value)  
yield\_corr2 = yield\_corr[order(-yield\_corr[,4]),]  
yield\_corr2

## Var1 Var2 value absvalue  
## 1 Yield Yield 1.00 1.00  
## 45 ManufacturingProcess32 Yield 0.61 0.61  
## 49 ManufacturingProcess36 Yield -0.53 0.53  
## 22 ManufacturingProcess09 Yield 0.50 0.50  
## 26 ManufacturingProcess13 Yield -0.50 0.50  
## 3 BiologicalMaterial02 Yield 0.48 0.48  
## 7 BiologicalMaterial06 Yield 0.48 0.48  
## 4 BiologicalMaterial03 Yield 0.45 0.45  
## 30 ManufacturingProcess17 Yield -0.43 0.43  
## 46 ManufacturingProcess33 Yield 0.42 0.42  
## 19 ManufacturingProcess06 Yield 0.39 0.39  
## 5 BiologicalMaterial04 Yield 0.38 0.38  
## 9 BiologicalMaterial08 Yield 0.38 0.38  
## 13 BiologicalMaterial12 Yield 0.37 0.37  
## 2 BiologicalMaterial01 Yield 0.36 0.36  
## 12 BiologicalMaterial11 Yield 0.35 0.35  
## 25 ManufacturingProcess12 Yield 0.35 0.35  
## 24 ManufacturingProcess11 Yield 0.32 0.32  
## 17 ManufacturingProcess04 Yield -0.26 0.26  
## 41 ManufacturingProcess28 Yield 0.26 0.26  
## 43 ManufacturingProcess30 Yield 0.23 0.23  
## 28 ManufacturingProcess15 Yield 0.22 0.22  
## 15 ManufacturingProcess02 Yield -0.21 0.21  
## 37 ManufacturingProcess24 Yield -0.21 0.21  
## 11 BiologicalMaterial10 Yield 0.20 0.20  
## 23 ManufacturingProcess10 Yield 0.20 0.20  
## 47 ManufacturingProcess34 Yield 0.17 0.17  
## 48 ManufacturingProcess35 Yield -0.17 0.17  
## 50 ManufacturingProcess37 Yield -0.16 0.16  
## 56 ManufacturingProcess43 Yield 0.16 0.16  
## 6 BiologicalMaterial05 Yield 0.15 0.15  
## 42 ManufacturingProcess29 Yield 0.15 0.15  
## 32 ManufacturingProcess19 Yield 0.13 0.13  
## 8 BiologicalMaterial07 Yield -0.11 0.11  
## 18 ManufacturingProcess05 Yield 0.11 0.11  
## 14 ManufacturingProcess01 Yield -0.10 0.10  
## 36 ManufacturingProcess23 Yield -0.10 0.10  
## 10 BiologicalMaterial09 Yield 0.09 0.09  
## 16 ManufacturingProcess03 Yield -0.09 0.09  
## 51 ManufacturingProcess38 Yield -0.09 0.09  
## 33 ManufacturingProcess20 Yield -0.07 0.07  
## 44 ManufacturingProcess31 Yield -0.07 0.07  
## 57 ManufacturingProcess44 Yield 0.07 0.07  
## 31 ManufacturingProcess18 Yield -0.06 0.06  
## 20 ManufacturingProcess07 Yield -0.04 0.04  
## 29 ManufacturingProcess16 Yield -0.04 0.04  
## 39 ManufacturingProcess26 Yield 0.04 0.04  
## 52 ManufacturingProcess39 Yield 0.04 0.04  
## 53 ManufacturingProcess40 Yield -0.04 0.04  
## 34 ManufacturingProcess21 Yield -0.03 0.03  
## 54 ManufacturingProcess41 Yield -0.03 0.03  
## 58 ManufacturingProcess45 Yield 0.03 0.03  
## 35 ManufacturingProcess22 Yield 0.02 0.02  
## 21 ManufacturingProcess08 Yield 0.01 0.01  
## 27 ManufacturingProcess14 Yield -0.01 0.01  
## 38 ManufacturingProcess25 Yield 0.01 0.01  
## 55 ManufacturingProcess42 Yield -0.01 0.01  
## 40 ManufacturingProcess27 Yield 0.00 0.00

library(AppliedPredictiveModeling)  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
  
  
#Create the regression model  
lmyield = lm(Yield~ManufacturingProcess32 +   
 ManufacturingProcess36 +   
 ManufacturingProcess09 +   
 ManufacturingProcess13 +   
 BiologicalMaterial02 +   
 BiologicalMaterial06 +   
 BiologicalMaterial03,   
 data=train)  
summary(lmyield)

##   
## Call:  
## lm(formula = Yield ~ ManufacturingProcess32 + ManufacturingProcess36 +   
## ManufacturingProcess09 + ManufacturingProcess13 + BiologicalMaterial02 +   
## BiologicalMaterial06 + BiologicalMaterial03, data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2.69278 -0.85654 -0.07033 0.75329 2.79231   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.498e+01 1.323e+01 1.132 0.259737   
## ManufacturingProcess32 1.562e-01 3.227e-02 4.840 3.79e-06 \*\*\*  
## ManufacturingProcess36 -3.344e+02 1.837e+02 -1.820 0.071161 .   
## ManufacturingProcess09 3.914e-01 1.121e-01 3.492 0.000665 \*\*\*  
## ManufacturingProcess13 -3.325e-01 1.705e-01 -1.950 0.053373 .   
## BiologicalMaterial02 -7.825e-02 8.475e-02 -0.923 0.357683   
## BiologicalMaterial06 8.893e-02 9.005e-02 0.988 0.325293   
## BiologicalMaterial03 9.045e-03 5.222e-02 0.173 0.862761   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.158 on 124 degrees of freedom  
## Multiple R-squared: 0.6395, Adjusted R-squared: 0.6191   
## F-statistic: 31.42 on 7 and 124 DF, p-value: < 2.2e-16

#### Response

For pre-processing we identified and removed variables with a near zero variance then converted our existing data frame into a long format. This process removed one variable, "BiologicalMaterial07." We then created a correlation matrix of the predictors with our response, ‘Yield,’ and down selected to reduce the data dimensions and complete our preprocessing steps.

After splitting the data 75% to training and 25% to our test set, we chose to build a linear regression model using the top 7 features that had the strongest correlations with ‘Yield.’ They were selected based on absolute correlation strength. A table of all variable correlations with Yield is shown for reference.

```{r, echo=FALSE}

flextable(yield\_corr2) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

### Those selected are shown in the model summary which had a multiple R2 value of `r lmmod.summary$r.squared` indicating the model captures roughly `r round(lmmod.summary$r.squared, 2)\*100`% of the variation in response. We note that, this is likely not the best model for prediction with this data and the only significant predictors (based on an alpha level of 0.05) were ManufacturingProcess32 and ManufacturingProcess09. However, this was a simple model choice we could gather information from and improve on as necessary since we were given the option to tune the model of our choice.

### Part D

#### Question

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?

#### Code

library(Metrics)  
#predict with the test data  
predict(lmyield, newdata = test, interval ='prediction')

## fit lwr upr  
## 149 37.41259 35.00037 39.82481  
## 52 41.84460 39.52770 44.16151  
## 4 42.24634 39.85691 44.63577  
## 127 40.07372 37.73076 42.41667  
## 11 41.29020 38.89561 43.68478  
## 102 37.74870 35.40103 40.09638  
## 94 39.30978 36.96410 41.65547  
## 18 39.49314 37.13267 41.85362  
## 51 42.73422 40.36832 45.10012  
## 157 38.29153 35.95422 40.62885  
## 53 42.14444 39.78547 44.50340  
## 131 40.48238 38.17011 42.79466  
## 39 42.24668 39.87978 44.61358  
## 56 41.10247 38.77185 43.43309  
## 85 40.69485 38.33934 43.05036  
## 105 39.15680 36.80813 41.50546  
## 176 41.80531 39.46396 44.14666  
## 141 39.22742 36.87256 41.58228  
## 7 42.46095 40.01785 44.90405  
## 79 39.78438 37.44680 42.12195  
## 124 41.25995 38.91519 43.60471  
## 22 42.17586 39.79299 44.55873  
## 62 39.62210 37.30479 41.93941  
## 160 39.48122 37.14975 41.81270  
## 37 42.21153 39.87214 44.55091  
## 113 39.45965 37.12248 41.79681  
## 114 39.29788 36.96211 41.63364  
## 154 39.14508 36.82980 41.46036  
## 140 38.53250 36.14940 40.91561  
## 122 38.17730 35.80653 40.54807  
## 142 39.86423 37.55825 42.17021  
## 164 39.60294 37.29613 41.90975  
## 128 39.85840 37.52754 42.18927  
## 109 39.48345 37.16804 41.79887  
## 104 38.50695 36.18421 40.82969  
## 63 39.93996 37.63438 42.24553  
## 144 40.17922 37.87046 42.48798  
## 137 39.11052 36.74819 41.47285  
## 45 42.12258 39.76390 44.48125  
## 136 39.01567 36.62883 41.40252  
## 59 41.80081 39.44977 44.15186  
## 14 40.48219 38.15210 42.81228  
## 1 38.54437 36.09784 40.99090  
## 42 40.62549 38.24608 43.00490

#Predict with the test data  
lmyield\_test = predict(lmyield, newdata = test, interval ='prediction')  
lmyield\_test\_df = data.frame(lmyield\_test)  
lmyield\_test\_df$actual = test$Yield  
lmyield\_test\_df

## fit lwr upr actual  
## 149 37.41259 35.00037 39.82481 38.63  
## 52 41.84460 39.52770 44.16151 40.14  
## 4 42.24634 39.85691 44.63577 41.42  
## 127 40.07372 37.73076 42.41667 42.28  
## 11 41.29020 38.89561 43.68478 42.04  
## 102 37.74870 35.40103 40.09638 37.86  
## 94 39.30978 36.96410 41.65547 39.91  
## 18 39.49314 37.13267 41.85362 40.14  
## 51 42.73422 40.36832 45.10012 41.12  
## 157 38.29153 35.95422 40.62885 37.14  
## 53 42.14444 39.78547 44.50340 42.69  
## 131 40.48238 38.17011 42.79466 40.89  
## 39 42.24668 39.87978 44.61358 42.23  
## 56 41.10247 38.77185 43.43309 39.40  
## 85 40.69485 38.33934 43.05036 40.91  
## 105 39.15680 36.80813 41.50546 38.60  
## 176 41.80531 39.46396 44.14666 39.49  
## 141 39.22742 36.87256 41.58228 38.66  
## 7 42.46095 40.01785 44.90405 43.12  
## 79 39.78438 37.44680 42.12195 40.41  
## 124 41.25995 38.91519 43.60471 43.42  
## 22 42.17586 39.79299 44.55873 41.18  
## 62 39.62210 37.30479 41.93941 38.20  
## 160 39.48122 37.14975 41.81270 37.86  
## 37 42.21153 39.87214 44.55091 42.46  
## 113 39.45965 37.12248 41.79681 39.96  
## 114 39.29788 36.96211 41.63364 39.79  
## 154 39.14508 36.82980 41.46036 37.51  
## 140 38.53250 36.14940 40.91561 39.77  
## 122 38.17730 35.80653 40.54807 40.66  
## 142 39.86423 37.55825 42.17021 40.31  
## 164 39.60294 37.29613 41.90975 38.67  
## 128 39.85840 37.52754 42.18927 41.62  
## 109 39.48345 37.16804 41.79887 39.51  
## 104 38.50695 36.18421 40.82969 37.87  
## 63 39.93996 37.63438 42.24553 38.70  
## 144 40.17922 37.87046 42.48798 40.64  
## 137 39.11052 36.74819 41.47285 37.39  
## 45 42.12258 39.76390 44.48125 43.33  
## 136 39.01567 36.62883 41.40252 38.03  
## 59 41.80081 39.44977 44.15186 42.31  
## 14 40.48219 38.15210 42.81228 40.28  
## 1 38.54437 36.09784 40.99090 38.00  
## 42 40.62549 38.24608 43.00490 42.07

lmyield\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
cat('RMSE of the test data for this model is', lmyield\_test\_rmse)

## RMSE of the test data for this model is 1.17824

#### Response

Our fit $R^2$ value is `r df.res$R2.fit[2]` which indicates our model explains about `r round(df.res$R2.fit[2],2)\*100`% on the test set. This is lower than optimal if we were making predictions based on it and our RMSE (root mean squared error) does not perform as well as it could either. Compared to the same metrics on the training dataset, the test data performed slightly poorer at prediction. We can trace these results back to the model selection, which, if reselected, would likely improve given the information we have gained from this model.

### Part E

#### Question

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

#### Response

As shown in the above table with rank, variable, and importance values, the manufacturing process takes the most importance. Process predictors are seemingly more important with our model. However, looking back at the correlations when down selecting for our model, manufacturing processes still dominated the list when considering all available variables.

Recall that since our model relied on the 7 strongest correlations with Yield, the number of variables used in the model were already reduced. We also note that only two of our variables had significance below an alpha level of 0.05. Both of those variables were also process predictors (ManufacturingProcess32 and ManufacturingProcess09).

### Part F

#### Question

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?

#### Response

We outline the relationships between each variable from the entire data set in the correlation plot where orange indicates a stronger positive relationship and blue indicates a stronger negative relationship.

Given our previous exploration of correlations, p-value significance levels, and feature fit with linear modeling, we can easily confirm that process predictors have more relevance in predicting yield. This bodes well as these features can be changes while biological ones cannot change. Simulations could be done using linear and other model types with modifications to focus on those predictors.

A good next step might be to build the model via forward stepwise or backward stepwise regression (or both). Training on some other models such as Random Forest, could also be used to identify which features are most relevant and expose their true nature when influencing yield. We note that prior to experimenting with process features value changes, it is difficult to know what will perform well.

## KJ 7.2

Friedman (1991) introduced several benchmark data sets created by simulation. One of these simulations used the following nonlinear equation to create data: y = 10 sin(πx1x2) + 20(x3 − 0.5)2 + 10x4 + 5x5 + N(0, σ2) 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: > library(mlbench) > 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. > trainingDatax) > ## Look at the data using > featurePlot(trainingDatay) > ## or other methods. > > ## 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) > testDatax)

### Part A

#### Question

Tune several models on these data. An example is shown in the code. Which models appear to give the best performance? Does MARS select the informative predictors (those named X1–X5)?

#### Code

library(mlbench)  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

##   
## Attaching package: 'caret'

## The following objects are masked from 'package:Metrics':  
##   
## precision, recall

library(AppliedPredictiveModeling)  
library(kernlab)

##   
## Attaching package: 'kernlab'

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

library(doParallel) # Used for computation

## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel

library(earth) # Package necessary for marsModel

## Loading required package: Formula

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos

registerDoParallel(cores=2)  
getDoParWorkers()

## [1] 2

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)  
## or other methods.  
## 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)  
knnModel <- train(x = trainingData$x, y = trainingData$y, method = "knn", preProc = c("center", "scale"), tuneLength = 10)  
knnPred <- predict(knnModel, newdata = testData$x)  
## The function 'postResample' can be used to get the test set   
## performance values  
postResample(pred = knnPred, obs = testData$y)

## RMSE Rsquared MAE   
## 3.2040595 0.6819919 2.5683461

svmModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "svmRadial",  
 tuneLength=10,  
 preProc = c("center", "scale"))  
svmPred <- predict(svmModel, newdata = testData$x)  
postResample(pred = svmPred, obs = testData$y)

## RMSE Rsquared MAE   
## 2.0864652 0.8236735 1.5854649

nnetGrid <- expand.grid(.decay=c(0, 0.01, 0.1, 0.5, 0.9),  
 .size=c(1, 10, 15, 20),  
 .bag=FALSE)  
nnetModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "avNNet",  
 tuneGrid = nnetGrid,  
 preProc = c("center", "scale"),  
 trace=FALSE,  
 linout=TRUE,  
 maxit=500)  
# Neural net may take several minutes  
nnetPred <- predict(nnetModel, newdata = testData$x)  
postResample(pred = nnetPred, obs = testData$y)

## RMSE Rsquared MAE   
## 1.894755 0.856176 1.441820

marsGrid <- expand.grid(.degree=1:2,  
 .nprune=2:20)  
marsModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "earth",  
 tuneGrid = marsGrid,  
 preProc = c("center", "scale"))  
marsPred <- predict(marsModel, newdata = testData$x)  
postResample(pred = marsPred, obs = testData$y)

## RMSE Rsquared MAE   
## 1.3227340 0.9291489 1.0524686

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.

svmModel

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 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:  
##   
## C RMSE Rsquared MAE   
## 0.25 2.545335 0.7804647 2.015121  
## 0.50 2.319786 0.7965148 1.830009  
## 1.00 2.188349 0.8119636 1.726027  
## 2.00 2.103655 0.8241314 1.655842  
## 4.00 2.066879 0.8294322 1.631051  
## 8.00 2.052681 0.8313929 1.623550  
## 16.00 2.049867 0.8318312 1.621820  
## 32.00 2.049867 0.8318312 1.621820  
## 64.00 2.049867 0.8318312 1.621820  
## 128.00 2.049867 0.8318312 1.621820  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.06802164  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were sigma = 0.06802164 and C = 16.

nnetModel

## Model Averaged Neural Network   
##   
## 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:  
##   
## decay size RMSE Rsquared MAE   
## 0.00 1 2.589902 0.7324403 2.014450  
## 0.00 10 3.082909 0.6581005 2.345346  
## 0.00 15 2.666684 0.7202739 2.135432  
## 0.00 20 2.634394 0.7273102 2.116201  
## 0.01 1 2.567190 0.7363972 1.993021  
## 0.01 10 2.714509 0.7118994 2.173845  
## 0.01 15 2.435366 0.7634558 1.935705  
## 0.01 20 2.346376 0.7801847 1.850974  
## 0.10 1 2.580129 0.7336990 2.000459  
## 0.10 10 2.528971 0.7492960 2.003431  
## 0.10 15 2.309856 0.7879857 1.823430  
## 0.10 20 2.289300 0.7922572 1.799799  
## 0.50 1 2.620985 0.7251648 2.034073  
## 0.50 10 2.389468 0.7734132 1.893293  
## 0.50 15 2.248817 0.7979988 1.778851  
## 0.50 20 2.257951 0.7973906 1.768133  
## 0.90 1 2.649162 0.7195330 2.057453  
## 0.90 10 2.339031 0.7803865 1.849270  
## 0.90 15 2.247236 0.7980673 1.774157  
## 0.90 20 2.248629 0.7989807 1.770371  
##   
## 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 = 15, decay = 0.9 and bag = FALSE.

marsModel

## Multivariate Adaptive Regression Spline   
##   
## 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:  
##   
## degree nprune RMSE Rsquared MAE   
## 1 2 4.420600 0.2049915 3.619816  
## 1 3 3.701246 0.4408654 2.951090  
## 1 4 2.799544 0.6800991 2.225051  
## 1 5 2.538474 0.7359840 2.028529  
## 1 6 2.392972 0.7670752 1.895457  
## 1 7 1.997878 0.8371559 1.571951  
## 1 8 1.876575 0.8570992 1.467108  
## 1 9 1.812964 0.8663208 1.420100  
## 1 10 1.782912 0.8704084 1.399319  
## 1 11 1.771684 0.8718386 1.386223  
## 1 12 1.787483 0.8701198 1.395825  
## 1 13 1.805686 0.8678252 1.407169  
## 1 14 1.804984 0.8683606 1.410113  
## 1 15 1.831547 0.8648770 1.433968  
## 1 16 1.839945 0.8637020 1.438650  
## 1 17 1.834362 0.8644089 1.434148  
## 1 18 1.834362 0.8644089 1.434148  
## 1 19 1.834362 0.8644089 1.434148  
## 1 20 1.834362 0.8644089 1.434148  
## 2 2 4.424916 0.2035789 3.628878  
## 2 3 3.680817 0.4479081 2.941723  
## 2 4 2.858745 0.6646861 2.265031  
## 2 5 2.589553 0.7239981 2.070119  
## 2 6 2.464649 0.7530842 1.944510  
## 2 7 2.090644 0.8201211 1.642564  
## 2 8 1.917597 0.8491333 1.508119  
## 2 9 1.808609 0.8641337 1.419806  
## 2 10 1.697999 0.8803917 1.328664  
## 2 11 1.600188 0.8932347 1.251957  
## 2 12 1.554919 0.8988995 1.222037  
## 2 13 1.482903 0.9082352 1.153521  
## 2 14 1.529860 0.9025187 1.186956  
## 2 15 1.551420 0.9004952 1.202656  
## 2 16 1.564789 0.8995960 1.211800  
## 2 17 1.555896 0.9007856 1.208746  
## 2 18 1.553024 0.9011837 1.207791  
## 2 19 1.566099 0.8996463 1.214404  
## 2 20 1.566099 0.8996463 1.214404  
##   
## 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.

postResample(pred = nnetPred, obs = testData$y)

## RMSE Rsquared MAE   
## 1.894755 0.856176 1.441820

postResample(pred = svmPred, obs = testData$y)

## RMSE Rsquared MAE   
## 2.0864652 0.8236735 1.5854649

postResample(pred = marsPred, obs = testData$y)

## RMSE Rsquared MAE   
## 1.3227340 0.9291489 1.0524686

varImp(marsModel)

## earth variable importance  
##   
## Overall  
## X1 100.00  
## X4 75.40  
## X2 49.00  
## X5 15.72  
## X3 0.00

#### Response

KNN, SVM, Neural Network, and MARS models were used for testing performances. Since the KNN model code was given as an example and did not perform as well as the others, we leave its performance statistics out.

Based on RMSE, the MARS model performs best. It also appears to select the informative predictors (X1-X5). A table of these statistics is provided to compare each model we created.

```{r, echo=FALSE}

flextable(modstat7.2) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

## KJ 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.

### Part A

#### Question

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

#### Code

# Recreate RMSE from 6.3  
# Part A load the data  
library(AppliedPredictiveModeling)  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
# Part B Fill in missing values with the median of each feature  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}  
# Part C Correlations and Modeling  
library(reshape2)  
library(dplyr)  
cormat <- round(cor(df),2)  
melted\_cormat = melt(cormat)  
melted\_cormat\_df = data.frame(melted\_cormat)  
#Filter to only Yield and sort in descending order  
yield\_corr <- melted\_cormat\_df %>%   
 filter(Var2 == "Yield")  
yield\_corr$absvalue = abs(yield\_corr$value)  
yield\_corr2 = yield\_corr[order(-yield\_corr[,4]),]  
yield\_corr2

## Var1 Var2 value absvalue  
## 1 Yield Yield 1.00 1.00  
## 45 ManufacturingProcess32 Yield 0.61 0.61  
## 49 ManufacturingProcess36 Yield -0.53 0.53  
## 22 ManufacturingProcess09 Yield 0.50 0.50  
## 26 ManufacturingProcess13 Yield -0.50 0.50  
## 3 BiologicalMaterial02 Yield 0.48 0.48  
## 7 BiologicalMaterial06 Yield 0.48 0.48  
## 4 BiologicalMaterial03 Yield 0.45 0.45  
## 30 ManufacturingProcess17 Yield -0.43 0.43  
## 46 ManufacturingProcess33 Yield 0.42 0.42  
## 19 ManufacturingProcess06 Yield 0.39 0.39  
## 5 BiologicalMaterial04 Yield 0.38 0.38  
## 9 BiologicalMaterial08 Yield 0.38 0.38  
## 13 BiologicalMaterial12 Yield 0.37 0.37  
## 2 BiologicalMaterial01 Yield 0.36 0.36  
## 12 BiologicalMaterial11 Yield 0.35 0.35  
## 25 ManufacturingProcess12 Yield 0.35 0.35  
## 24 ManufacturingProcess11 Yield 0.32 0.32  
## 17 ManufacturingProcess04 Yield -0.26 0.26  
## 41 ManufacturingProcess28 Yield 0.26 0.26  
## 43 ManufacturingProcess30 Yield 0.23 0.23  
## 28 ManufacturingProcess15 Yield 0.22 0.22  
## 15 ManufacturingProcess02 Yield -0.21 0.21  
## 37 ManufacturingProcess24 Yield -0.21 0.21  
## 11 BiologicalMaterial10 Yield 0.20 0.20  
## 23 ManufacturingProcess10 Yield 0.20 0.20  
## 47 ManufacturingProcess34 Yield 0.17 0.17  
## 48 ManufacturingProcess35 Yield -0.17 0.17  
## 50 ManufacturingProcess37 Yield -0.16 0.16  
## 56 ManufacturingProcess43 Yield 0.16 0.16  
## 6 BiologicalMaterial05 Yield 0.15 0.15  
## 42 ManufacturingProcess29 Yield 0.15 0.15  
## 32 ManufacturingProcess19 Yield 0.13 0.13  
## 8 BiologicalMaterial07 Yield -0.11 0.11  
## 18 ManufacturingProcess05 Yield 0.11 0.11  
## 14 ManufacturingProcess01 Yield -0.10 0.10  
## 36 ManufacturingProcess23 Yield -0.10 0.10  
## 10 BiologicalMaterial09 Yield 0.09 0.09  
## 16 ManufacturingProcess03 Yield -0.09 0.09  
## 51 ManufacturingProcess38 Yield -0.09 0.09  
## 33 ManufacturingProcess20 Yield -0.07 0.07  
## 44 ManufacturingProcess31 Yield -0.07 0.07  
## 57 ManufacturingProcess44 Yield 0.07 0.07  
## 31 ManufacturingProcess18 Yield -0.06 0.06  
## 20 ManufacturingProcess07 Yield -0.04 0.04  
## 29 ManufacturingProcess16 Yield -0.04 0.04  
## 39 ManufacturingProcess26 Yield 0.04 0.04  
## 52 ManufacturingProcess39 Yield 0.04 0.04  
## 53 ManufacturingProcess40 Yield -0.04 0.04  
## 34 ManufacturingProcess21 Yield -0.03 0.03  
## 54 ManufacturingProcess41 Yield -0.03 0.03  
## 58 ManufacturingProcess45 Yield 0.03 0.03  
## 35 ManufacturingProcess22 Yield 0.02 0.02  
## 21 ManufacturingProcess08 Yield 0.01 0.01  
## 27 ManufacturingProcess14 Yield -0.01 0.01  
## 38 ManufacturingProcess25 Yield 0.01 0.01  
## 55 ManufacturingProcess42 Yield -0.01 0.01  
## 40 ManufacturingProcess27 Yield 0.00 0.00

library(AppliedPredictiveModeling)  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
#Create the regression model  
lmyield = lm(Yield~ManufacturingProcess32 +   
 ManufacturingProcess36 +   
 ManufacturingProcess09 +   
 ManufacturingProcess13 +   
 BiologicalMaterial02 +   
 BiologicalMaterial06 +   
 BiologicalMaterial03,   
 data=train)  
summary(lmyield)

##   
## Call:  
## lm(formula = Yield ~ ManufacturingProcess32 + ManufacturingProcess36 +   
## ManufacturingProcess09 + ManufacturingProcess13 + BiologicalMaterial02 +   
## BiologicalMaterial06 + BiologicalMaterial03, data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2.44421 -0.90710 -0.01637 0.65844 2.77625   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 8.74867 13.49126 0.648 0.517880   
## ManufacturingProcess32 0.15921 0.03200 4.975 2.13e-06 \*\*\*  
## ManufacturingProcess36 -227.74633 176.47679 -1.291 0.199272   
## ManufacturingProcess09 0.39310 0.11507 3.416 0.000859 \*\*\*  
## ManufacturingProcess13 -0.25506 0.17520 -1.456 0.147966   
## BiologicalMaterial02 -0.03588 0.08290 -0.433 0.665897   
## BiologicalMaterial06 0.03966 0.09129 0.434 0.664707   
## BiologicalMaterial03 0.02330 0.05627 0.414 0.679492   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.142 on 124 degrees of freedom  
## Multiple R-squared: 0.584, Adjusted R-squared: 0.5605   
## F-statistic: 24.86 on 7 and 124 DF, p-value: < 2.2e-16

# Part D Predictions  
library(Metrics)  
# predict with the test data  
predict(lmyield, newdata = test, interval ='prediction')

## fit lwr upr  
## 25 42.77604 40.43479 45.11728  
## 151 38.43567 36.13067 40.74067  
## 176 41.75110 39.43371 44.06850  
## 159 39.30670 37.02850 41.58489  
## 126 39.59217 37.30427 41.88007  
## 24 41.71571 39.34341 44.08801  
## 103 38.33210 36.01535 40.64885  
## 3 42.52655 40.11831 44.93479  
## 48 42.76404 40.42088 45.10720  
## 112 39.53533 37.20728 41.86338  
## 4 42.14915 39.75946 44.53885  
## 154 39.17472 36.89197 41.45746  
## 12 42.12787 39.75432 44.50142  
## 132 40.04697 37.76775 42.32619  
## 23 41.26694 38.91400 43.61988  
## 77 39.96947 37.67576 42.26318  
## 36 39.83317 37.53358 42.13276  
## 32 42.07901 39.76120 44.39683  
## 175 41.10639 38.79195 43.42082  
## 27 36.73048 34.17654 39.28441  
## 46 42.12946 39.79717 44.46174  
## 69 39.34328 37.06874 41.61782  
## 17 41.26832 38.90222 43.63442  
## 26 37.31403 34.81260 39.81545  
## 82 41.50135 39.06888 43.93382  
## 156 38.11272 35.81537 40.41008  
## 107 39.50866 37.23090 41.78642  
## 71 39.24312 36.94497 41.54126  
## 63 39.93102 37.65794 42.20410  
## 169 40.50364 38.16061 42.84667  
## 140 38.48902 36.13284 40.84520  
## 49 43.33629 40.92890 45.74369  
## 53 42.06928 39.73544 44.40312  
## 2 42.45565 40.06938 44.84192  
## 81 39.84315 37.52872 42.15759  
## 144 40.17583 37.89852 42.45314  
## 67 41.64302 39.31128 43.97477  
## 134 39.58431 37.27683 41.89178  
## 9 41.37090 39.03220 43.70960  
## 148 38.04294 35.71787 40.36800  
## 113 39.59029 37.27450 41.90607  
## 29 38.11138 35.52380 40.69896  
## 43 41.83860 39.53510 44.14210  
## 56 41.13524 38.83282 43.43766

# Predict with the test data  
lmyield\_test = predict(lmyield, newdata = test, interval ='prediction')  
lmyield\_test\_df = data.frame(lmyield\_test)  
lmyield\_test\_df$actual = test$Yield  
lmyield\_test\_df

## fit lwr upr actual  
## 25 42.77604 40.43479 45.11728 43.38  
## 151 38.43567 36.13067 40.74067 40.96  
## 176 41.75110 39.43371 44.06850 39.49  
## 159 39.30670 37.02850 41.58489 38.03  
## 126 39.59217 37.30427 41.88007 41.31  
## 24 41.71571 39.34341 44.08801 41.89  
## 103 38.33210 36.01535 40.64885 38.05  
## 3 42.52655 40.11831 44.93479 42.03  
## 48 42.76404 40.42088 45.10720 43.84  
## 112 39.53533 37.20728 41.86338 40.19  
## 4 42.14915 39.75946 44.53885 41.42  
## 154 39.17472 36.89197 41.45746 37.51  
## 12 42.12787 39.75432 44.50142 42.68  
## 132 40.04697 37.76775 42.32619 40.82  
## 23 41.26694 38.91400 43.61988 40.70  
## 77 39.96947 37.67576 42.26318 38.73  
## 36 39.83317 37.53358 42.13276 40.87  
## 32 42.07901 39.76120 44.39683 41.87  
## 175 41.10639 38.79195 43.42082 38.48  
## 27 36.73048 34.17654 39.28441 35.25  
## 46 42.12946 39.79717 44.46174 42.61  
## 69 39.34328 37.06874 41.61782 39.71  
## 17 41.26832 38.90222 43.63442 40.89  
## 26 37.31403 34.81260 39.81545 36.83  
## 82 41.50135 39.06888 43.93382 41.25  
## 156 38.11272 35.81537 40.41008 36.77  
## 107 39.50866 37.23090 41.78642 39.42  
## 71 39.24312 36.94497 41.54126 39.16  
## 63 39.93102 37.65794 42.20410 38.70  
## 169 40.50364 38.16061 42.84667 38.90  
## 140 38.48902 36.13284 40.84520 39.77  
## 49 43.33629 40.92890 45.74369 46.34  
## 53 42.06928 39.73544 44.40312 42.69  
## 2 42.45565 40.06938 44.84192 42.44  
## 81 39.84315 37.52872 42.15759 39.79  
## 144 40.17583 37.89852 42.45314 40.64  
## 67 41.64302 39.31128 43.97477 41.96  
## 134 39.58431 37.27683 41.89178 38.05  
## 9 41.37090 39.03220 43.70960 41.49  
## 148 38.04294 35.71787 40.36800 39.14  
## 113 39.59029 37.27450 41.90607 39.96  
## 29 38.11138 35.52380 40.69896 38.52  
## 43 41.83860 39.53510 44.14210 44.35  
## 56 41.13524 38.83282 43.43766 39.40

lmyield\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
cat('RMSE of the test data for this model is', lmyield\_test\_rmse)

## RMSE of the test data for this model is 1.223971

# Apply 6.3 to 7.5   
library(caret)  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
#Create the train x and train y datasets for HW 7.5  
trainx = train[,2:ncol(train)]  
trainy = train$Yield  
  
testx = test[,2:ncol(test)]  
testy = test$Yield  
#Using KNN for HW 7.5  
#Train KNN  
knnModel <- train(x=trainx, y=trainy)  
knnModel

## Random Forest   
##   
## 132 samples  
## 57 predictor  
##   
## No pre-processing  
## Resampling: Bootstrapped (25 reps)   
## Summary of sample sizes: 132, 132, 132, 132, 132, 132, ...   
## Resampling results across tuning parameters:  
##   
## mtry RMSE Rsquared MAE   
## 2 1.370286 0.5748113 1.114728  
## 29 1.291664 0.5664502 1.010097  
## 57 1.328372 0.5327052 1.026103  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was mtry = 29.

#Example from the book on how to predict  
knnPred <- predict(knnModel, newdata = testx)  
## The function 'postResample' can be used to get the test set  
## performance values  
postResample(pred = knnPred, obs = testy)

## RMSE Rsquared MAE   
## 0.9497665 0.6896668 0.7368882

#Using MARS for HW7.5  
#Train using MARS  
marsFit <- earth(trainx, trainy)  
summary(marsFit)

## Call: earth(x=trainx, y=trainy)  
##   
## coefficients  
## (Intercept) 39.144063  
## h(72.41-BiologicalMaterial03) -0.089877  
## h(12.5-ManufacturingProcess01) -0.167684  
## h(209.8-ManufacturingProcess06) -0.196802  
## h(43.84-ManufacturingProcess09) -0.456702  
## h(ManufacturingProcess09-43.84) 0.301157  
## h(33.1-ManufacturingProcess13) 2.004453  
## h(10.2-ManufacturingProcess28) 0.053345  
## h(ManufacturingProcess32-152) 0.242609  
## h(61-ManufacturingProcess33) 0.481826  
## h(7.2-ManufacturingProcess39) -0.240470  
## h(ManufacturingProcess39-7.2) -3.885665  
##   
## Selected 12 of 21 terms, and 9 of 57 predictors  
## Termination condition: RSq changed by less than 0.001 at 21 terms  
## Importance: ManufacturingProcess32, ManufacturingProcess09, ...  
## Number of terms at each degree of interaction: 1 11 (additive model)  
## GCV 1.189166 RSS 107.034 GRSq 0.6765851 RSq 0.7760916

#MARS tune  
# Define the candidate models to test  
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)  
# Fix the seed so that the results can be reproduced  
set.seed(100)  
marsTuned <- train(trainx, trainy,method = "earth",tuneGrid = marsGrid, trControl = trainControl(method = "cv"))  
marsTuned

## Multivariate Adaptive Regression Spline   
##   
## 132 samples  
## 57 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 119, 119, 118, 120, 119, 118, ...   
## Resampling results across tuning parameters:  
##   
## degree nprune RMSE Rsquared MAE   
## 1 2 1.451069 0.4676867 1.1540743  
## 1 3 1.335116 0.5220707 1.0931425  
## 1 4 1.343203 0.5263327 1.1104647  
## 1 5 1.355265 0.5268058 1.0990902  
## 1 6 1.273949 0.5691816 1.0242302  
## 1 7 1.299613 0.5537436 1.0819112  
## 1 8 1.361062 0.5159438 1.0961474  
## 1 9 1.401224 0.5065660 1.1232654  
## 1 10 1.402384 0.5169186 1.1194640  
## 1 11 1.369493 0.5315924 1.1137267  
## 1 12 1.330221 0.5403815 1.0767529  
## 1 13 1.339242 0.5414844 1.0750374  
## 1 14 1.346520 0.5393709 1.0860846  
## 1 15 1.350232 0.5354554 1.0929352  
## 1 16 1.350232 0.5354554 1.0929352  
## 1 17 1.350232 0.5354554 1.0929352  
## 1 18 1.350232 0.5354554 1.0929352  
## 1 19 1.350232 0.5354554 1.0929352  
## 1 20 1.350232 0.5354554 1.0929352  
## 1 21 1.350232 0.5354554 1.0929352  
## 1 22 1.350232 0.5354554 1.0929352  
## 1 23 1.350232 0.5354554 1.0929352  
## 1 24 1.350232 0.5354554 1.0929352  
## 1 25 1.350232 0.5354554 1.0929352  
## 1 26 1.350232 0.5354554 1.0929352  
## 1 27 1.350232 0.5354554 1.0929352  
## 1 28 1.350232 0.5354554 1.0929352  
## 1 29 1.350232 0.5354554 1.0929352  
## 1 30 1.350232 0.5354554 1.0929352  
## 1 31 1.350232 0.5354554 1.0929352  
## 1 32 1.350232 0.5354554 1.0929352  
## 1 33 1.350232 0.5354554 1.0929352  
## 1 34 1.350232 0.5354554 1.0929352  
## 1 35 1.350232 0.5354554 1.0929352  
## 1 36 1.350232 0.5354554 1.0929352  
## 1 37 1.350232 0.5354554 1.0929352  
## 1 38 1.350232 0.5354554 1.0929352  
## 2 2 1.451069 0.4676867 1.1540743  
## 2 3 1.310292 0.5202345 1.0623547  
## 2 4 1.185512 0.6219954 0.9644920  
## 2 5 1.210826 0.6160088 0.9936183  
## 2 6 1.279618 0.5917242 1.0220261  
## 2 7 1.320625 0.5845782 1.0475047  
## 2 8 1.285528 0.6031614 1.0112410  
## 2 9 1.255311 0.6229948 0.9789426  
## 2 10 1.249870 0.6230238 0.9793693  
## 2 11 1.290423 0.6089043 1.0146814  
## 2 12 1.276335 0.6230513 0.9981833  
## 2 13 1.299153 0.6111072 1.0212064  
## 2 14 1.312451 0.6106052 1.0344145  
## 2 15 1.347846 0.6049178 1.0430479  
## 2 16 1.325749 0.6157096 1.0184699  
## 2 17 1.325561 0.6166835 1.0157942  
## 2 18 1.309769 0.6224005 1.0084643  
## 2 19 1.337595 0.6193371 1.0304649  
## 2 20 1.364375 0.6147098 1.0571589  
## 2 21 1.382851 0.6110499 1.0667885  
## 2 22 1.382851 0.6110499 1.0667885  
## 2 23 1.379967 0.6122970 1.0820284  
## 2 24 1.379967 0.6122970 1.0820284  
## 2 25 1.379967 0.6122970 1.0820284  
## 2 26 1.379967 0.6122970 1.0820284  
## 2 27 1.379967 0.6122970 1.0820284  
## 2 28 1.379967 0.6122970 1.0820284  
## 2 29 1.379967 0.6122970 1.0820284  
## 2 30 1.379967 0.6122970 1.0820284  
## 2 31 1.379967 0.6122970 1.0820284  
## 2 32 1.379967 0.6122970 1.0820284  
## 2 33 1.379967 0.6122970 1.0820284  
## 2 34 1.379967 0.6122970 1.0820284  
## 2 35 1.379967 0.6122970 1.0820284  
## 2 36 1.379967 0.6122970 1.0820284  
## 2 37 1.379967 0.6122970 1.0820284  
## 2 38 1.379967 0.6122970 1.0820284  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were nprune = 4 and degree = 2.

#Predict with MARS  
head(predict(marsTuned, testx))

## y  
## [1,] 39.83914  
## [2,] 38.13659  
## [3,] 39.84109  
## [4,] 39.68153  
## [5,] 39.30739  
## [6,] 39.85167

#Relevant features with MARS  
varImp(marsTuned)

## earth variable importance  
##   
## Overall  
## ManufacturingProcess32 100  
## ManufacturingProcess09 0

#### Response

KNN, SVM, and MARS models were trained, summarized, tuned, and tested with the chemical data. The SVM model performs the best in its R squared value and has a lower MAE. For this we would use the SVM model. However, the MARS model produced a lower RMSE in our prediction test. See the model test results table based on predicted data for more descriptive statistics.

### Part B

#### Question

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?

#### Code

Done. [missing code to show below - fill in]

#### Response

The process variables dominate the list regardless of how we review the models. The top ten most important predictors are majority process based. If we assume the SVM is our best and review its predictors we can produce the following rank for each predictor with its importance score:

```{r}

flextable(impsvm.tbl[1:10,]) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

Alternatively, we can perform the same calculations on the MARS model if it were selected as our best and there would only be 3 predictors with ManufacturingProcess32 of most importance. Our MARS ranking would look shorter.

```{r}

flextable(impmar.tbl) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

We note that the methods used to determine the optimal nonlinear regression model are different than those of the optimal linear model. For example, our varImp function in the caret package of R does not contain a method for selecting model type when evaluating predictor importance. Instead, it seems to filter the variables through regression, fit a LOESS line from predictor to outcome, and calculate an R squared term for each predictor. It then orders the predictors by R squared value in descending order, arranging the highest (and most important) predictors at the top. This is not the same for all model types.

### 

### Part C

#### Question

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?

#### Code

Done.

#### Response

Scatterplots with LOESS and linear estimates are fit to our 10 most important predictors from the SVM model to gain an understanding of their relationships with Yield as it is observed. We notice that ManufacturingProcess32 has the clearest linear relationship with yield and the process predictors each seem to have a single optimal value for maximizing yield (beyond which there are diminishing returns). Biological predictors remain less predictable to our model. Thus, we can say part of our intuition is confirmed based on our exploration of the data. However, our radial SVM model transformed the data such that we are not able to visualize its relationship clearly in two-dimensional space. Thus, the initial observations before modeling provide the best visualization to gain well-rounded understanding.

## KJ 8.1

8.1. Recreate the simulated data from Exercise 7.2:

library(mlbench) set.seed(200) simulated <- mlbench.friedman1(200, sd = 1) simulated <- cbind(simulatedy) simulated <- as.data.frame(simulated) colnames(simulated)[ncol(simulated)] <- “y”

library(mlbench)  
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

## The following object is masked from 'package:dplyr':  
##   
## combine

library(caret)  
library(partykit)

## Loading required package: grid

## Loading required package: libcoin

## Loading required package: mvtnorm

library(dplyr)  
library(gbm)

## Loaded gbm 2.1.8

library(Cubist)  
set.seed(200)  
simulated <- mlbench.friedman1(200, sd = 1)  
simulated <- cbind(simulated$x, simulated$y)  
simulated <- as.data.frame(simulated)  
colnames(simulated)[ncol(simulated)] <- "y"  
model1 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp1 <- varImp(model1, scale = FALSE)  
rfImp1

## Overall  
## V1 8.732235404  
## V2 6.415369387  
## V3 0.763591825  
## V4 7.615118809  
## V5 2.023524577  
## V6 0.165111172  
## V7 -0.005961659  
## V8 -0.166362581  
## V9 -0.095292651  
## V10 -0.074944788

simulated$duplicate1 <- simulated$V1 + rnorm(200) \* .1  
cor(simulated$duplicate1, simulated$V1)

## [1] 0.9460206

model2 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp2 <- varImp(model2, scale = FALSE)  
rfImp2

## Overall  
## V1 5.69119973  
## V2 6.06896061  
## V3 0.62970218  
## V4 7.04752238  
## V5 1.87238438  
## V6 0.13569065  
## V7 -0.01345645  
## V8 -0.04370565  
## V9 0.00840438  
## V10 0.02894814  
## duplicate1 4.28331581

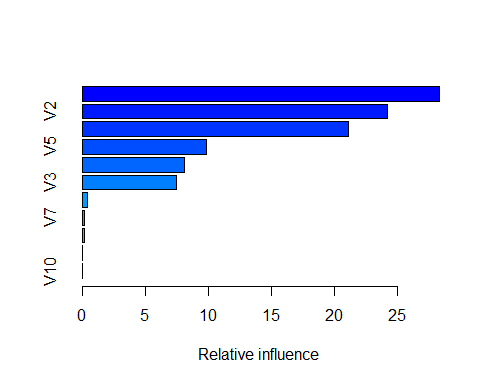
cforest\_model <- cforest(y ~ ., data=simulated)  
# Unconditional importance measure  
varimp(cforest\_model) %>% sort(decreasing = T)

## V4 V1 V2 duplicate1 V5 V3   
## 6.54418720 6.14253842 6.10410006 5.73199392 2.07799547 0.10632869   
## V7 V9 V8 V6 V10   
## 0.05872350 -0.08030916 -0.08585961 -0.11606490 -0.14555774

varimp(cforest\_model, conditional=T) %>% sort(decreasing = T)

## V4 V2 V1 duplicate1 V5 V3   
## 5.87536300 5.19402511 3.22747234 2.62605554 1.29939244 0.07065979   
## V9 V7 V6 V10 V8   
## -0.05486725 -0.09302146 -0.14118239 -0.18846005 -0.30875505

gbm\_Model <- gbm(y ~ ., data=simulated, distribution='gaussian')  
summary(gbm\_Model)



## var rel.inf  
## V4 V4 28.3814249  
## V2 V2 24.2747829  
## V1 V1 21.1520418  
## V5 V5 9.8965623  
## duplicate1 duplicate1 8.1127217  
## V3 V3 7.4639180  
## V6 V6 0.3890084  
## V7 V7 0.1717731  
## V9 V9 0.1577668  
## V8 V8 0.0000000  
## V10 V10 0.0000000

cubistModel <- cubist(x=simulated[,-(ncol(simulated)-1)], y=simulated$y, committees=100)  
varImp(cubistModel)

## Overall  
## V3 43.5  
## V1 52.5  
## V2 59.5  
## duplicate1 27.5  
## V4 46.0  
## V8 4.0  
## V5 27.0  
## V6 10.0  
## V10 1.0  
## V7 0.0  
## V9 0.0

### Part A

#### Question

Fit a random forest model to all of the predictors, then estimate the variable importance scores. Did the random forest model significantly use the uninformative predictors (V6 – V10)?

#### Code

library(randomForest)  
library(caret)  
model1 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp1 <- varImp(model1, scale = FALSE)  
rfImp1

## Overall  
## V1 5.61129716  
## V2 6.25100292  
## V3 0.48920645  
## V4 6.93545032  
## V5 1.89349345  
## V6 0.10311608  
## V7 -0.02429715  
## V8 -0.14558489  
## V9 -0.02242304  
## V10 0.03740779  
## duplicate1 4.36798537

#### Response

Given our ranked table of importance by predictor, V6 – V10 are near zero or negative indicating their insignificance in the random forest model.

### Part B

#### Question

Now add an additional predictor that is highly correlated with one of the informative predictors. Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictor that is also highly correlated with V1?

#### Code

model2 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp2 <- varImp(model2, scale = FALSE)  
rfImp2

## Overall  
## V1 5.370727680  
## V2 6.359790192  
## V3 0.513709259  
## V4 6.987407442  
## V5 1.965321937  
## V6 0.064864835  
## V7 -0.001954341  
## V8 -0.050730616  
## V9 -0.038538162  
## V10 0.033235844  
## duplicate1 4.575077699

Response

A new table of ranked predictors by importance shows that when adding a predictor that is highly correlated with one of the informative predictors the importance of the predictor is split across the correlated variables. For example, adding a predictor highly correlated with V1 dropped its importance from about 9 to roughly 5 and the remaining importance was reassigned to the highly correlated ‘duplicate1’ variable. Predictors V4 and V2 held their importance the same. By maintaining the same level while V1’s importance dropped, both V4 and V2 became the two most importance predictors.

### Part C

#### Question

Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can calculate predictor importance. The conditional argument of that function toggles between the traditional importance measure and the modified version described in Strobl et al. (2007). Do these importances show the same pattern as the traditional random forest model?

#### Code

library(partykit)  
library(dplyr)  
cforest\_model <- cforest(y ~ ., data=simulated)  
# Unconditional importance measure  
varimp(cforest\_model) %>% sort(decreasing = T)

## V4 duplicate1 V1 V2 V5   
## 6.4940046552 6.0948594529 5.8994610286 5.8023438061 1.8662231965   
## V3 V9 V10 V6 V7   
## 0.1340853644 0.1216699774 -0.0001368413 -0.0221268311 -0.0618254673   
## V8   
## -0.1351737391

varimp(cforest\_model, conditional=T) %>% sort(decreasing = T)

## V4 V2 V1 duplicate1 V5 V3   
## 5.72582613 4.91183522 3.00077657 2.73102425 1.24394321 0.02362531   
## V10 V9 V6 V7 V8   
## -0.05284822 -0.12571833 -0.13369852 -0.26651287 -0.30868876

#### Response

The uninformative predictors V6 - V10 are still rated with low importance. Predictor ‘duplicate1’ (that is highly correlated with V1) was reduced in importance from the varimp() function. All other importance values are also reduced. From this, we can conclude the patterns between traditional and modified measures of importance are very similar in order and arrangement, but the modified measure reduces predictor importance to a smaller scale. When importance is already small, this may reorder the insignificant predictors as V6-V10 have been.

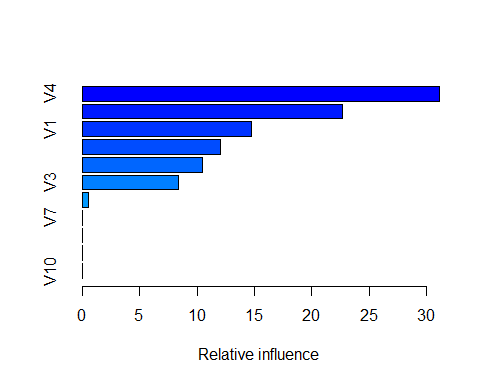
### Part D

#### Question

Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur?

#### Code

library(gbm)  
library(Cubist)  
gbm\_Model <- gbm(y ~ ., data=simulated, distribution='gaussian')  
summary(gbm\_Model)



## var rel.inf  
## V4 V4 31.1424277  
## V2 V2 22.6896980  
## V1 V1 14.7824367  
## duplicate1 duplicate1 12.0474916  
## V5 V5 10.4422291  
## V3 V3 8.3638973  
## V6 V6 0.5318196  
## V7 V7 0.0000000  
## V8 V8 0.0000000  
## V9 V9 0.0000000  
## V10 V10 0.0000000

cubistModel <- cubist(x=simulated[,-(ncol(simulated)-1)], y=simulated$y, committees=100)  
varImp(cubistModel)

## Overall  
## V3 43.5  
## V1 52.5  
## V2 59.5  
## duplicate1 27.5  
## V4 46.0  
## V8 4.0  
## V5 27.0  
## V6 10.0  
## V10 1.0  
## V7 0.0  
## V9 0.0

#### Response

The top-rated predictor is V4 in a GBM model and V1 in a Cubist model. Predictors V6 - V10 remain very low in importance. The predictor ‘duplicate1’ scored a 0 for importance in the Cubist model but higher in GBM. We review the patterns through two tables with the predictors from GBM first.

```{r, echo=FALSE}

flextable(gbm.tbl) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

For GBM compared to random forest, the scale of importance to the model has changed. In GBM no value is negative. The lowest bound is 0. GBM is also a measure of relative influence, not strictly importance which seems to follow a more exponential scale from most to least important predictors. We also review the Cubist model’s predictors.

```{r, echo=FALSE}

flextable(cube.varimp.tbl) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

For Cubist model the change in vector magnitude from least to most important is even greater than GBM (and far from any random forest). It is also bounded by 0 at its lowest like GBM but the orders are slightly different. Predictor ‘duplicate1’ is ranked 5th in both models but V1 ranks higher in Cubist.

Based on the observations above, the pattern is different across the models.

## KJ 8.2

Use a simulation to show tree bias with different granularities.

### Part A

#### Question

Fit a random forest model to all of the predictors, then estimate the variable importance scores:

#### Code

library(caret)  
library(rpart)  
set.seed(755)  
X1 <- rep(1:2, each=100)  
Y <- X1 + rnorm(600, mean=2, sd=4)  
X2 <- rnorm(600, mean=2, sd=4)  
simData <- data.frame(Y=Y, X1=X1, X2=X2)  
fit <- rpart(Y ~ ., data = simData)  
varImp(fit)

## Overall  
## X1 0.021731921  
## X2 0.005446745

#### Response

The simulation was created with two variables. Each was assigned a mean of 2 and standard deviation of 4. The two variables differ in variance.

X1 -> Lesser variance (Sequence of values)

X2 -> Higher Variance (Gaussian Random value)

Target Variable Y -> X1 + X2

A regression tree is fitted using recursive partitioning and importance calculated with the caret package’s varImp() method. We can see simulation ‘X1’ has greater importance than ‘X2.’ The variance also differs with ‘X2’ being over 56 times greater than ‘X1.’ The difference between two variables is significant and it shows the tree bias between two variables.

## KJ 8.3

In stochastic gradient boosting the bagging fraction and learning rate will govern the construction of the trees as they are guided by the gradient. Although the optimal values of these parameters should be obtained through the tuning process, it is helpful to understand how the magnitudes of these parameters affect magnitudes of variable importance. Figure 8.24 provides the variable importance plots for boosting using two extreme values for the bagging fraction (0.1 and 0.9) and the learning rate (0.1 and 0.9) for the solubility data. The left-hand plot has both parameters set to 0.1, and the right-hand plot has both set to 0.9:

### Diagram Description automatically generated with low confidence

### Part A

#### Question

Why does the model on the right focus its importance on just the first few of predictors, whereas the model on the left spreads importance across more predictors?

#### Response

There could be two reasons for the model on the right spreads importance on the first few predictors and the model on the left adds importance to few more predictors. Bragging Fraction : It represents the data usage in each iteration of trees. The left hand plot has a bragging fraction of 0.1 which is low and only 10% of data is used for random sampling. Whereas , the right hand plot has a bragging fraction of 0.9 which is large and 90% of data is used on each iteration which is almost the full dataset. Since the full data set is approximately used by the model on the right plot, only few predictors got importance but the left plot used partial dataset to get importance over few more predictors. Learning Rate : It means a higher number of predictions are added to the model output. Since the right hand plot has more predictions the correlation is more and hence only the first few predictors were considered significant.

There could be two reasons the model on the right focuses its importance on the first few predictors while the model on the left spreads its importance out. These are outlined below:

Bragging Fraction – it represent the data usage in each interation of trees. The plot on the left has a bragging fraction of 0.1 which is low and only 105 of the data is used for random sampling. Whereas, the plot on the right has a bragging fraction of 0.9 which is quite large comparatively. It indicates 90% of the data is used on each iteration which is alsmot the full data set. Since almost the full data set is used (by the model on the right), a handful of predictors are able to pick up more importance. This concentrates the importance as shown in the plot on the right. The opposite occurs in the plot on the left with where only 10% of the data set is used.

Learning Rate – it means a higher number of predictions are added to the model output. Since the plot on the right has more predictions, the correlation is greater. Thus, only the first few predictors were considered significant.

### Part B

#### Question

Which model do you think would be more predictive of other samples?

Response

Since the bragging fraction and learning rate can control overfitting of the model, they are crucial in selecting the more predictive model. If we are trying to cover more predictions for unseen samples, then a smaller bragging fraction and lower learning rate should be more predictive than one with a larger fraction and higher rate. This is because it includes a higher proportion of possible sample predictions. Thus, the left plot would probably be more predictive of other samples. However, there is a point that one must make a bias-variance compromise to remain usefully predictive.

Part C

Question

How would increasing interaction depth affect the slope of predictor importance for either model in Fig. 8.24?

#### Code

library(gbm)  
library(AppliedPredictiveModeling)  
data(solubility)  
grid1 <- expand.grid(n.trees=100, interaction.depth=1, shrinkage=0.1, n.minobsinnode=10)  
gbm1 <- train(x = solTrainXtrans, y = solTrainY, method = 'gbm', tuneGrid = grid1, verbose = FALSE)  
grid2 <- expand.grid(n.trees=100, interaction.depth=10, shrinkage=0.1, n.minobsinnode=10)  
gbm2 <- train(x = solTrainXtrans, y = solTrainY, method = 'gbm', tuneGrid = grid2, verbose = FALSE)  
varImp(gbm1)

## gbm variable importance  
##   
## only 20 most important variables shown (out of 228)  
##   
## Overall  
## NumCarbon 100.000  
## MolWeight 63.707  
## SurfaceArea2 36.382  
## NumAromaticBonds 24.449  
## NumChlorine 19.016  
## SurfaceArea1 16.448  
## HydrophilicFactor 11.640  
## NumHalogen 6.092  
## NumMultBonds 6.049  
## FP112 4.867  
## FP172 4.762  
## NumRotBonds 2.703  
## NumNonHAtoms 2.070  
## FP204 1.905  
## FP135 1.750  
## FP147 1.683  
## FP059 1.522  
## FP161 1.479  
## NumHydrogen 1.435  
## FP142 1.255

varImp(gbm2)

## gbm variable importance  
##   
## only 20 most important variables shown (out of 228)  
##   
## Overall  
## NumCarbon 100.000  
## MolWeight 50.699  
## SurfaceArea1 32.876  
## SurfaceArea2 24.533  
## HydrophilicFactor 16.340  
## NumNonHAtoms 10.719  
## NumChlorine 6.765  
## NumMultBonds 5.529  
## NumAromaticBonds 5.165  
## NumHydrogen 4.443  
## NumRotBonds 3.507  
## NumHalogen 2.596  
## FP112 2.359  
## FP092 2.351  
## NumNonHBonds 2.321  
## FP072 2.147  
## NumAtoms 2.088  
## FP142 1.947  
## FP172 1.789  
## NumOxygen 1.712

#### Response

Increasing interaction depth improves predictor importance significantly. Two tables were created from two GBM models measuring importance of each predictor. They show the first 10 predictors and their rank by importance in descending order. The only difference between the models is interaction depth. The first contains an interaction depth of 1 and is shown:

```{r, echo=F}

flextable(gbm1.tbl[1:10,]) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

The second contains an interaction depth of 10. Notice the difference in importance scores and ranked order of importance compared to the first table.

```{r, echo=F}

flextable(gbm2.tbl[1:10,]) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

## The order of predictors changed (see ranks 3 – 10) and the importance decreased quicker with greater interaction depth. With the interaction depth at 10, the importance spread across more predictors shrinking the importance values for the first 10 predictors in the second table.

## KJ 8.7

Refer to Exercises 6.3 and 7.5 which describe a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several tree-based models:

### Part A

#### Question

Which tree-based regression model gives the optimal resampling and test set performance?

#### Code

# Code from exercise 6.3 and 7.5  
library(AppliedPredictiveModeling)  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
sum(is.na(train))

## [1] 0

sum(is.na(test))

## [1] 0

# Create the Random Forest Tree  
rf\_model = randomForest(Yield~ManufacturingProcess32 + ManufacturingProcess36 + ManufacturingProcess09 + ManufacturingProcess13 + BiologicalMaterial02 + BiologicalMaterial06 + BiologicalMaterial03, data=train)

library(Metrics)  
library(AppliedPredictiveModeling)  
library(reshape2)  
library(dplyr)  
# Apply 6.3 and 7.5   
# Predict with the test data  
rf\_test = predict(rf\_model, newdata = test)  
rf\_test\_df = data.frame(rf\_test)  
rf\_test\_df$actual = test$Yield  
rf\_test\_df

## rf\_test actual  
## 47 42.98636 42.96  
## 66 41.79396 42.03  
## 136 38.92333 38.03  
## 45 42.08091 43.33  
## 137 38.77698 37.39  
## 145 38.57768 38.60  
## 126 39.67213 41.31  
## 4 41.96159 41.42  
## 61 40.55274 40.57  
## 85 41.01977 40.91  
## 94 39.87375 39.91  
## 92 39.73507 39.17  
## 68 40.57660 41.85  
## 27 38.02320 35.25  
## 132 39.82845 40.82  
## 29 38.84805 38.52  
## 140 39.34096 39.77  
## 115 40.62755 41.86  
## 108 39.64786 39.75  
## 86 39.83864 38.99  
## 12 42.07851 42.68  
## 109 39.39215 39.51  
## 50 41.33978 39.74  
## 35 39.26226 39.65  
## 170 40.79951 39.62  
## 160 39.15347 37.86  
## 111 40.25377 40.38  
## 119 38.96651 40.19  
## 158 37.81245 37.73  
## 128 40.11336 41.62  
## 62 39.79973 38.20  
## 32 42.77332 41.87  
## 10 42.24008 42.45  
## 173 40.08241 39.68  
## 7 42.46666 43.12  
## 159 38.80983 38.03  
## 1 38.90723 38.00  
## 93 39.65095 39.98  
## 123 39.59328 42.58  
## 127 40.22364 42.28  
## 133 39.39976 39.77  
## 171 40.72203 39.77  
## 38 42.38178 42.66  
## 129 40.72799 42.73

rf\_test\_rmse = rmse(rf\_test\_df$actual, rf\_test\_df$rf\_test)  
cat('RMSE of the test data for this model is', rf\_test\_rmse)

## RMSE of the test data for this model is 1.110375

#Create the ctree   
ctree\_model = cforest(Yield~ManufacturingProcess32 + ManufacturingProcess36 + ManufacturingProcess09 + ManufacturingProcess13 + BiologicalMaterial02 + BiologicalMaterial06 + BiologicalMaterial03, data=train)  
#Predict with the test data  
ctree\_test = predict(ctree\_model, newdata = test)  
ctree\_test\_df = data.frame(ctree\_test)  
ctree\_test\_df$actual = test$Yield  
ctree\_test\_df

## ctree\_test actual  
## 47 42.28989 42.96  
## 66 41.41717 42.03  
## 136 39.14437 38.03  
## 45 42.15557 43.33  
## 137 38.94091 37.39  
## 145 38.81602 38.60  
## 126 39.53135 41.31  
## 4 41.49024 41.42  
## 61 40.48009 40.57  
## 85 40.82203 40.91  
## 94 39.56414 39.91  
## 92 39.61206 39.17  
## 68 40.75850 41.85  
## 27 39.33028 35.25  
## 132 39.49058 40.82  
## 29 39.30559 38.52  
## 140 39.22680 39.77  
## 115 40.79161 41.86  
## 108 39.80814 39.75  
## 86 40.14055 38.99  
## 12 41.76597 42.68  
## 109 39.39905 39.51  
## 50 41.14741 39.74  
## 35 39.38000 39.65  
## 170 41.10210 39.62  
## 160 39.17765 37.86  
## 111 40.55180 40.38  
## 119 38.41791 40.19  
## 158 38.37865 37.73  
## 128 39.74319 41.62  
## 62 39.78404 38.20  
## 32 42.19623 41.87  
## 10 42.02730 42.45  
## 173 39.93615 39.68  
## 7 41.28074 43.12  
## 159 38.80325 38.03  
## 1 38.91631 38.00  
## 93 39.34694 39.98  
## 123 39.52460 42.58  
## 127 39.83227 42.28  
## 133 39.46444 39.77  
## 171 40.84534 39.77  
## 38 42.33944 42.66  
## 129 40.44045 42.73

# Uses lmyield from 6.3   
# Recreate lmyield and RMSE from 6.3   
# Part A load the data  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
# Part B Fill in missing values with the median of each feature  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}  
# Part C Correlations and Modeling  
cormat <- round(cor(df),2)  
melted\_cormat = melt(cormat)  
melted\_cormat\_df = data.frame(melted\_cormat)  
#Filter to only Yield and sort in descending order  
yield\_corr <- melted\_cormat\_df %>%   
 filter(Var2 == "Yield")  
yield\_corr$absvalue = abs(yield\_corr$value)  
yield\_corr2 = yield\_corr[order(-yield\_corr[,4]),]  
yield\_corr2

## Var1 Var2 value absvalue  
## 1 Yield Yield 1.00 1.00  
## 45 ManufacturingProcess32 Yield 0.61 0.61  
## 49 ManufacturingProcess36 Yield -0.53 0.53  
## 22 ManufacturingProcess09 Yield 0.50 0.50  
## 26 ManufacturingProcess13 Yield -0.50 0.50  
## 3 BiologicalMaterial02 Yield 0.48 0.48  
## 7 BiologicalMaterial06 Yield 0.48 0.48  
## 4 BiologicalMaterial03 Yield 0.45 0.45  
## 30 ManufacturingProcess17 Yield -0.43 0.43  
## 46 ManufacturingProcess33 Yield 0.42 0.42  
## 19 ManufacturingProcess06 Yield 0.39 0.39  
## 5 BiologicalMaterial04 Yield 0.38 0.38  
## 9 BiologicalMaterial08 Yield 0.38 0.38  
## 13 BiologicalMaterial12 Yield 0.37 0.37  
## 2 BiologicalMaterial01 Yield 0.36 0.36  
## 12 BiologicalMaterial11 Yield 0.35 0.35  
## 25 ManufacturingProcess12 Yield 0.35 0.35  
## 24 ManufacturingProcess11 Yield 0.32 0.32  
## 17 ManufacturingProcess04 Yield -0.26 0.26  
## 41 ManufacturingProcess28 Yield 0.26 0.26  
## 43 ManufacturingProcess30 Yield 0.23 0.23  
## 28 ManufacturingProcess15 Yield 0.22 0.22  
## 15 ManufacturingProcess02 Yield -0.21 0.21  
## 37 ManufacturingProcess24 Yield -0.21 0.21  
## 11 BiologicalMaterial10 Yield 0.20 0.20  
## 23 ManufacturingProcess10 Yield 0.20 0.20  
## 47 ManufacturingProcess34 Yield 0.17 0.17  
## 48 ManufacturingProcess35 Yield -0.17 0.17  
## 50 ManufacturingProcess37 Yield -0.16 0.16  
## 56 ManufacturingProcess43 Yield 0.16 0.16  
## 6 BiologicalMaterial05 Yield 0.15 0.15  
## 42 ManufacturingProcess29 Yield 0.15 0.15  
## 32 ManufacturingProcess19 Yield 0.13 0.13  
## 8 BiologicalMaterial07 Yield -0.11 0.11  
## 18 ManufacturingProcess05 Yield 0.11 0.11  
## 14 ManufacturingProcess01 Yield -0.10 0.10  
## 36 ManufacturingProcess23 Yield -0.10 0.10  
## 10 BiologicalMaterial09 Yield 0.09 0.09  
## 16 ManufacturingProcess03 Yield -0.09 0.09  
## 51 ManufacturingProcess38 Yield -0.09 0.09  
## 33 ManufacturingProcess20 Yield -0.07 0.07  
## 44 ManufacturingProcess31 Yield -0.07 0.07  
## 57 ManufacturingProcess44 Yield 0.07 0.07  
## 31 ManufacturingProcess18 Yield -0.06 0.06  
## 20 ManufacturingProcess07 Yield -0.04 0.04  
## 29 ManufacturingProcess16 Yield -0.04 0.04  
## 39 ManufacturingProcess26 Yield 0.04 0.04  
## 52 ManufacturingProcess39 Yield 0.04 0.04  
## 53 ManufacturingProcess40 Yield -0.04 0.04  
## 34 ManufacturingProcess21 Yield -0.03 0.03  
## 54 ManufacturingProcess41 Yield -0.03 0.03  
## 58 ManufacturingProcess45 Yield 0.03 0.03  
## 35 ManufacturingProcess22 Yield 0.02 0.02  
## 21 ManufacturingProcess08 Yield 0.01 0.01  
## 27 ManufacturingProcess14 Yield -0.01 0.01  
## 38 ManufacturingProcess25 Yield 0.01 0.01  
## 55 ManufacturingProcess42 Yield -0.01 0.01  
## 40 ManufacturingProcess27 Yield 0.00 0.00

#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
#Create the regression model  
lmyield = lm(Yield~ManufacturingProcess32 +   
 ManufacturingProcess36 +   
 ManufacturingProcess09 +   
 ManufacturingProcess13 +   
 BiologicalMaterial02 +   
 BiologicalMaterial06 +   
 BiologicalMaterial03,   
 data=train)  
summary(lmyield)

##   
## Call:  
## lm(formula = Yield ~ ManufacturingProcess32 + ManufacturingProcess36 +   
## ManufacturingProcess09 + ManufacturingProcess13 + BiologicalMaterial02 +   
## BiologicalMaterial06 + BiologicalMaterial03, data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2.74186 -0.91157 -0.02729 0.71934 2.72149   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 13.87794 13.43406 1.033 0.30359   
## ManufacturingProcess32 0.16791 0.03250 5.167 9.23e-07 \*\*\*  
## ManufacturingProcess36 -191.44541 185.29139 -1.033 0.30352   
## ManufacturingProcess09 0.32642 0.11618 2.810 0.00576 \*\*   
## ManufacturingProcess13 -0.37703 0.16825 -2.241 0.02681 \*   
## BiologicalMaterial02 -0.05752 0.08619 -0.667 0.50579   
## BiologicalMaterial06 0.07249 0.09588 0.756 0.45105   
## BiologicalMaterial03 0.01901 0.05289 0.359 0.71988   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.188 on 124 degrees of freedom  
## Multiple R-squared: 0.6165, Adjusted R-squared: 0.5949   
## F-statistic: 28.48 on 7 and 124 DF, p-value: < 2.2e-16

# Part D Predictions  
# predict with the test data  
predict(lmyield, newdata = test, interval ='prediction')

## fit lwr upr  
## 94 39.40259 36.99495 41.81023  
## 69 39.39373 37.02643 41.76104  
## 147 39.66549 37.24445 42.08653  
## 30 39.99762 37.40831 42.58692  
## 159 39.40176 37.03023 41.77330  
## 52 41.82547 39.44726 44.20368  
## 111 40.10204 37.70291 42.50116  
## 47 43.10386 40.63810 45.56962  
## 71 39.26863 36.87605 41.66120  
## 156 38.19271 35.80226 40.58316  
## 78 40.64560 38.25508 43.03612  
## 160 39.48916 37.09385 41.88447  
## 176 41.90002 39.50223 44.29782  
## 106 38.90199 36.52444 41.27953  
## 173 40.62668 38.23919 43.01418  
## 99 38.70108 36.27384 41.12832  
## 154 39.22447 36.84800 41.60094  
## 167 38.28864 35.89583 40.68146  
## 164 39.64903 37.28353 42.01454  
## 40 41.68893 39.29059 44.08728  
## 12 42.07301 39.61385 44.53218  
## 3 42.69938 40.22004 45.17872  
## 124 41.15356 38.74636 43.56077  
## 104 38.65211 36.26573 41.03849  
## 126 39.64163 37.26095 42.02231  
## 144 40.25072 37.88268 42.61876  
## 108 39.97289 37.58096 42.36482  
## 92 39.84025 37.42152 42.25899  
## 98 40.23735 37.85713 42.61757  
## 166 38.56350 36.16976 40.95724  
## 27 36.81000 34.20256 39.41745  
## 96 39.81920 37.43585 42.20254  
## 122 38.18705 35.76419 40.60991  
## 145 39.00106 36.60664 41.39547  
## 81 39.85696 37.45729 42.25663  
## 87 40.06448 37.67849 42.45047  
## 95 40.97164 38.56187 43.38141  
## 54 41.43320 39.00914 43.85725  
## 107 39.50659 37.13653 41.87666  
## 37 42.35203 39.95650 44.74755  
## 115 40.81618 38.43052 43.20183  
## 46 42.32936 39.91454 44.74418  
## 152 39.29922 36.92484 41.67360  
## 4 42.31460 39.85336 44.77584

# Predict with the test data  
lmyield\_test = predict(lmyield, newdata = test, interval ='prediction')  
lmyield\_test\_df = data.frame(lmyield\_test)  
lmyield\_test\_df$actual = test$Yield  
lmyield\_test\_df

## fit lwr upr actual  
## 94 39.40259 36.99495 41.81023 39.91  
## 69 39.39373 37.02643 41.76104 39.71  
## 147 39.66549 37.24445 42.08653 40.10  
## 30 39.99762 37.40831 42.58692 38.35  
## 159 39.40176 37.03023 41.77330 38.03  
## 52 41.82547 39.44726 44.20368 40.14  
## 111 40.10204 37.70291 42.50116 40.38  
## 47 43.10386 40.63810 45.56962 42.96  
## 71 39.26863 36.87605 41.66120 39.16  
## 156 38.19271 35.80226 40.58316 36.77  
## 78 40.64560 38.25508 43.03612 38.95  
## 160 39.48916 37.09385 41.88447 37.86  
## 176 41.90002 39.50223 44.29782 39.49  
## 106 38.90199 36.52444 41.27953 38.44  
## 173 40.62668 38.23919 43.01418 39.68  
## 99 38.70108 36.27384 41.12832 37.94  
## 154 39.22447 36.84800 41.60094 37.51  
## 167 38.28864 35.89583 40.68146 38.82  
## 164 39.64903 37.28353 42.01454 38.67  
## 40 41.68893 39.29059 44.08728 41.43  
## 12 42.07301 39.61385 44.53218 42.68  
## 3 42.69938 40.22004 45.17872 42.03  
## 124 41.15356 38.74636 43.56077 43.42  
## 104 38.65211 36.26573 41.03849 37.87  
## 126 39.64163 37.26095 42.02231 41.31  
## 144 40.25072 37.88268 42.61876 40.64  
## 108 39.97289 37.58096 42.36482 39.75  
## 92 39.84025 37.42152 42.25899 39.17  
## 98 40.23735 37.85713 42.61757 40.81  
## 166 38.56350 36.16976 40.95724 39.15  
## 27 36.81000 34.20256 39.41745 35.25  
## 96 39.81920 37.43585 42.20254 39.86  
## 122 38.18705 35.76419 40.60991 40.66  
## 145 39.00106 36.60664 41.39547 38.60  
## 81 39.85696 37.45729 42.25663 39.79  
## 87 40.06448 37.67849 42.45047 38.81  
## 95 40.97164 38.56187 43.38141 40.77  
## 54 41.43320 39.00914 43.85725 40.15  
## 107 39.50659 37.13653 41.87666 39.42  
## 37 42.35203 39.95650 44.74755 42.46  
## 115 40.81618 38.43052 43.20183 41.86  
## 46 42.32936 39.91454 44.74418 42.61  
## 152 39.29922 36.92484 41.67360 37.89  
## 4 42.31460 39.85336 44.77584 41.42

lmyield\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
cat('RMSE of the test data for this model is', lmyield\_test\_rmse)

## RMSE of the test data for this model is 1.107951

# Create ctree with lmyield  
ctree\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
cat('RMSE of the test data for this model is', ctree\_test\_rmse)

## RMSE of the test data for this model is 1.107951

#### Response

Comparing random forest and ctree models, we found that the random forest model performed better with a lower root mean squared error (RMSE), lower mean absolute error (MAE) and higher coefficient of determination (Rsquared). Determining the optimal model often depends on its application and which source of error you attempt to reduce the most. For prediction in general, a lower RMSE would probably fare better than Rsquared and take priority over MAE.

Comparing SingleTree, RandomForest, CTree, Cubist, and CForest models, we found that the cubist model performed best due to lower root mean squared error RMSE, higher coefficient of determination (Rsquared), and lowest mean absolute error MAE. Although this was a single evaluation. Future tests could switch these results as we would expect with prediction of unknown data sources.

Additionally, determining the optimal model often depends on its application and which source of error you attempt to reduce the most. For prediction in general, a lower RMSE would probably fare better than Rsquared and take priority over MAE but it all depends on the problem and circumstances.

### Part B

#### Question

Which predictors are most important in the optimal tree-based regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and nonlinear models?

#### Code

[missing code - fill in to demonstrate response]

#### Response

The most important predictors in every model run thus far, including these two tree-based models, have been process predictors. The process variables held the majority of spots in the top 10 most important predictors in almost every model run thus far as well. However, some biological predictors creeped into the top 10 with our ctree and random forest models. Take a look at the ranked 2, 3, 5, and 9 spots in the table of importance from the random forest model:

```{r, echo = FALSE}

flextable(rf[1:10,]) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

Now, look at ranked spots 2, 3, 5, 9, and 10 in our ctree model. Importance values have also changed scale, maxing out near 100, while our ctree maximum importance value was just over 1.

```{r, echo = FALSE}

flextable(ct[1:10,]) %>%

theme\_vanilla() %>%

set\_table\_properties(layout = "autofit")

```

In any case, the top 10 predictors in linear and nonlinear models imply the same general rule, that manufacturing processes (especially ManufacturingProcess32, ManufacturingProcess13, and ManufacturingProcess09) are the most important predictors of Yield. If one desires to maximize Yield, it would be best advised to improve these processes.

### Part C

#### Question

Plot the optimal single tree with the distribution of yield in the terminal nodes. Does this view of the data provide additional knowledge about the biological or process predictors and their relationship with yield? #### Code

[missing a plot chunk for visual interpretation]

#### Response

## Through the hierarchical structure shown in this plot of a single tree model, we can easily determine the most important predictor (ManufacturingProcess32) and its relationship with Yield through nodes (or limbs since we are using tree references). It also shows us how importance it gets distributed to the next two most important predictors in this data set. We could continue this through all the branches of a single model to see where all predictors are ranked.

## For example, the single split in this model uses data from ManufacturingProcess32, which we suspected was the most significant influencer of Yield from our previous models. From this view we also notice that Yield and ManufacturingProcess32 are linearly related. An increase in one should result in an increase in the other. Since this is single tree model, we know that values are assigned by computing the mean of the group and splitting it by our single most important predictor, which is almost certainly, ManufacturingProcess32. Market Basket Analysis

I am assigning one simple problem on market basket analysis / recommender systems. Imagine 10000 receipts sitting on your table. Each receipt represents a transaction with items that were purchased. The receipt is a representation of stuff that went into a customer’s basket – and therefore ‘Market Basket Analysis’. That is exactly what the Groceries Data Set contains: a collection of receipts with each line representing 1 receipt and the items purchased. Each line is called a transaction and each column in a row represents an item. Here is the dataset = GroceryDataSet.csv (comma separated file) Your assignment is to use R to mine the data for association rules. You should report support, confidence and lift and your top 10 rules by lift.

#### Code

library(arules)

## Loading required package: Matrix

##   
## Attaching package: 'arules'

## The following object is masked from 'package:kernlab':  
##   
## size

## The following object is masked from 'package:dplyr':  
##   
## recode

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

library(arulesViz)  
library(RColorBrewer)  
library(visNetwork)  
library(igraph)

##   
## Attaching package: 'igraph'

## The following object is masked from 'package:arules':  
##   
## union

## The following objects are masked from 'package:dplyr':  
##   
## as\_data\_frame, groups, union

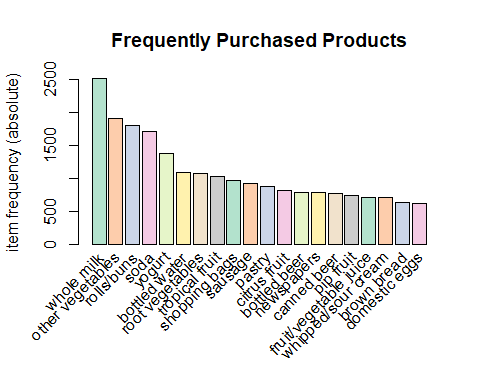
## The following objects are masked from 'package:stats':  
##   
## decompose, spectrum

## The following object is masked from 'package:base':  
##   
## union

df <- read.csv(  
"https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/GroceryDataSet.csv")  
df\_sparse <- read.transactions(  
"https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/GroceryDataSet.csv",  
 format="basket",sep=",")  
summary(df\_sparse)

## transactions as itemMatrix in sparse format with  
## 9835 rows (elements/itemsets/transactions) and  
## 169 columns (items) and a density of 0.02609146   
##   
## most frequent items:  
## whole milk other vegetables rolls/buns soda   
## 2513 1903 1809 1715   
## yogurt (Other)   
## 1372 34055   
##   
## element (itemset/transaction) length distribution:  
## sizes  
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16   
## 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55 46   
## 17 18 19 20 21 22 23 24 26 27 28 29 32   
## 29 14 14 9 11 4 6 1 1 1 1 3 1   
##   
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 2.000 3.000 4.409 6.000 32.000   
##   
## includes extended item information - examples:  
## labels  
## 1 abrasive cleaner  
## 2 artif. sweetener  
## 3 baby cosmetics

itemFrequencyPlot(df\_sparse,topN=20,type="absolute",col=brewer.pal(8,'Pastel2'), main="Frequently Purchased Products")



association.rules <- apriori(df\_sparse, parameter = list(supp=0.004, conf=0.3))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.3 0.1 1 none FALSE TRUE 5 0.004 1  
## maxlen target ext  
## 10 rules TRUE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 39   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].  
## sorting and recoding items ... [126 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 5 done [0.00s].  
## writing ... [735 rule(s)] done [0.00s].  
## creating S4 object ... done [0.00s].

length(association.rules)

## [1] 735

inspect(sort(association.rules, by = 'lift')[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {liquor} => {bottled beer} 0.004677173 0.4220183 0.011082867 5.240594 46  
## [2] {berries,   
## whole milk} => {whipped/sour cream} 0.004270463 0.3620690 0.011794611 5.050990 42  
## [3] {herbs,   
## whole milk} => {root vegetables} 0.004168785 0.5394737 0.007727504 4.949369 41  
## [4] {citrus fruit,   
## other vegetables,   
## tropical fruit} => {root vegetables} 0.004473818 0.4943820 0.009049314 4.535678 44  
## [5] {other vegetables,   
## root vegetables,   
## tropical fruit} => {citrus fruit} 0.004473818 0.3636364 0.012302999 4.393567 44  
## [6] {citrus fruit,   
## other vegetables,   
## root vegetables} => {tropical fruit} 0.004473818 0.4313725 0.010371124 4.110997 44  
## [7] {citrus fruit,   
## other vegetables,   
## whole milk} => {root vegetables} 0.005795628 0.4453125 0.013014743 4.085493 57  
## [8] {citrus fruit,   
## root vegetables,   
## tropical fruit} => {other vegetables} 0.004473818 0.7857143 0.005693950 4.060694 44  
## [9] {herbs} => {root vegetables} 0.007015760 0.4312500 0.016268429 3.956477 69  
## [10] {tropical fruit,   
## whipped/sour cream,   
## whole milk} => {yogurt} 0.004372140 0.5512821 0.007930859 3.951792 43

subset.rules <- which(colSums(is.subset(association.rules, association.rules)) > 1) # get subset rules in vector  
length(subset.rules)

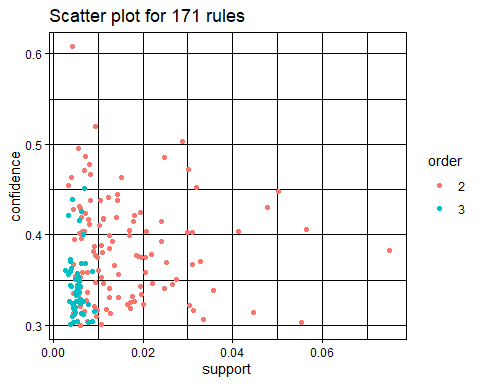
## [1] 564

subset.association.rules. <- association.rules[-subset.rules] # remove subset rules.  
inspect(sort(subset.association.rules., by = 'lift')[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {liquor} => {bottled beer} 0.004677173 0.4220183 0.011082867 5.240594 46  
## [2] {herbs} => {root vegetables} 0.007015760 0.4312500 0.016268429 3.956477 69  
## [3] {citrus fruit,   
## pip fruit} => {tropical fruit} 0.005592272 0.4044118 0.013828165 3.854060 55  
## [4] {pip fruit,   
## root vegetables} => {tropical fruit} 0.005287239 0.3398693 0.015556685 3.238967 52  
## [5] {tropical fruit,   
## whipped/sour cream} => {yogurt} 0.006202339 0.4485294 0.013828165 3.215224 61  
## [6] {margarine,   
## tropical fruit} => {yogurt} 0.004067107 0.4347826 0.009354347 3.116681 40  
## [7] {citrus fruit,   
## root vegetables} => {tropical fruit} 0.005693950 0.3218391 0.017691917 3.067139 56  
## [8] {pip fruit,   
## rolls/buns} => {tropical fruit} 0.004473818 0.3211679 0.013929842 3.060742 44  
## [9] {beef} => {root vegetables} 0.017386884 0.3313953 0.052465684 3.040367 171  
## [10] {tropical fruit,   
## whipped/sour cream} => {root vegetables} 0.004575496 0.3308824 0.013828165 3.035660 45

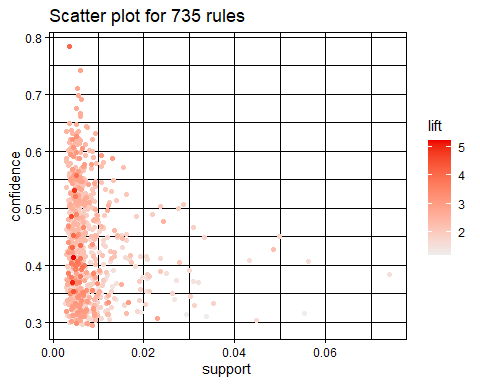
plot(subset.association.rules.,method="two-key plot")

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.



plot(association.rules)

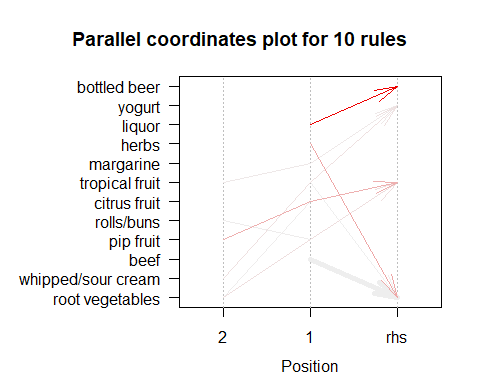
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.



top10subRules <- head(subset.association.rules., n = 10, by = "lift")  
plot(top10subRules, method = "graph", engine = "htmlwidget")

## PhantomJS not found. You can install it with webshot::install\_phantomjs(). If it is installed, please make sure the phantomjs executable can be found via the PATH variable.

subRules2<-head(subset.association.rules., n=10, by="lift")  
plot(subRules2, method="paracoord")



#### Response

Our association rules mining was achieved using apriori algorithm functions available in the arules package, which we use throughout this analysis. Association rules mining is a two-step process:

1. Frequent Itemset Generation
2. Rules Generation

Once the rules are generated, the top 10 rules were classified based on the output. To get there we loaded and summarized the data set as a sparce matrix. We then found the most frequently purchased products. The top 20 are shown in our code. We created our choice of minimum support to help classify the data. This formula is:

(Products purchased at least 5 times a day)/total number of transactions Supp = (6\*7)/nrow(df) = 0.004

Through trial and error on different combinations of items, we found that the confidence of 0.3 generated a decent number of rules for our combination. This produced a total of 735 rules and the top 10 rules are displayed as our “Parallel coordinates plot for 10 rules.” It appears the highest association is created for liquor and bottled beer with the life value of 5.24.

Going further, we cleaned up the redundant rules by subseting and down selecting our combination. The resultant highest association did not change but the other rules did. The results are shown in the “Parallel coordinates plot for 10 rules.” The most dependent purchases remained liquor and bottled beer with the life value of 5.24.