pH Prediction (Data 624 Project 2)

Heather Geiger

Libraries and set seed

```
library(ggplot2)
library(tidyr)
library(dplyr)
library(gridExtra)
library(impute)
library(caret)
library(pls)
```

Read in data.

Read in CSV files, which are just the CSV versions of the original Excel sheets.

```
training <- read.csv("pH_prediction_training_data.csv",header=TRUE,stringsAsFactors=F
ALSE)
test <- read.csv("pH_prediction_test_data.csv",header=TRUE,stringsAsFactors=FALSE)
```

Set aside target (PH) from training into a separate variable. Remove entirely from test.

Then, combine training and test into one data frame.

```
training_target <- training$PH

training <- training[,setdiff(colnames(training),"PH")]
test <- test[,colnames(training)]

alldata <- data.frame(rbind(training,test),
    Data = rep(c("Training","Test"),times=c(nrow(training),nrow(test))),
    stringsAsFactors=FALSE)
```

Data exploration summary

Data exploration is detailed more completely in another document.

Summary of findings:

- Most variables (including brand code) are missing at least some values.
- Most observations are only missing at most one or two variables. Although, there are a few that are missing a good proportion of variables.

- In Pressure.Setpoint, the vast majority of values are multiples of 2. So the ones that are between multiples (45.2, 46.4, 46.6, and 46.8) are unusual.
- Similarly in Bowl.Setpoint, 122, 126, and 134 are unusual in being not multiples of 10.
- Hyd.Pressure 1, 2, and 3 have a very large number of observations equal to exactly 0. We also find that Hyd.Pressure2 has an unusual number of observations equal to 0.2, while Hyd.Pressure3 has an unusual number of observations equal to -1.2.
- Mnf.Flow has over 600 observations *each* equal to -100 and -100.2. Then it also has 92 observations equal to 0.2.
- Many observations are bimodal or multimodal, and/or have varying degrees of skew.
- We find many of the predictors are correlated to some degree. Most notable of these were Density and Balling, where nearly all observations (except one in the test data) could be modeled by either Density = (.40 x Balling) + .32 or by Density = (.40 x Balling) - .17.

Based on this exploration, I propose the following manual data transformations in addition to the typical automated processing.

- Hyd.Pressure1 - Include a dummy variable for whether or not the variable is exactly equal to 0.
- Hyd.Pressure2 - Include three dummy variables.
    1. Equal to 0 vs. not
    2. Equal to 0.2 vs. not
    3. Equal to 0 or 0.2 vs. equal to neither
- Hyd.Pressure3 - Include three dummy variables similar to Hyd.Pressure2 (but for -1.2 instead of 0.2).
- Mnf.Flow - Include the following dummy variables.
    1. Equal to -100 vs. not
    2. Equal to -100.2 vs. not
    3. Equal to -100 or -100.2 vs. not
    4. Equal to 0.2 vs. not
    5. Equal to -100, -100.2, or 0.2 vs. not
- Pressure.Setpoint and Bowl.Setpoint - For the most part, round to the nearest appropriate multiple. Except, even though 126 is technically closer to 130 than 120, let's round down for that one since 120 is a much more common value here.
- Remove Density. Instead have Balling, plus a new variable with the intercept of the equation to get density (either -0.17 or +0.32). Change to NA then impute the new variable for the one test observation that did not fit either linear correlation.

Data transformation

Some initial minor transformations

First, let's save a copy of the data before transformation in case we want to refer to it later.

```
alldata_original <- alldata
```

Add dummy variables for Brand.Code.

```
brand_code_dummy_vars <- data.frame(model.matrix(~Brand.Code,data=alldata)[,2:5],stri
ngsAsFactors=FALSE)
brand_code_dummy_vars[rowSums(brand_code_dummy_vars) == 0,] <- NA
brand_code_dummy_vars <- brand_code_dummy_vars[,2:4]

alldata <- data.frame(alldata[,setdiff(colnames(alldata),"Brand.Code")],
    brand_code_dummy_vars,stringsAsFactors=FALSE)
```

Need to remove all categorical variables that have not been converted to dummy before run impute.knn.

```
training_vs_test_vector <- alldata$Data
alldata <- alldata[,setdiff(colnames(alldata),"Data")]
```

If Density is >= 0.4 x Balling, set variable Density_model_intercept_positive equal to 1. Otherwise, set equal to 0. Except, set NA for the observation where neither linear model fit well.

Then, remove Density.

```
alldata <- data.frame(alldata,
    Density_model_intercept_positive = ifelse(alldata$Density >= (0.4*alldata$Balling
),1,0),
    stringsAsFactors=FALSE)

alldata[which(alldata$Balling > 3 & alldata$Density < 0.5),"Density_model_intercept_p
ositive"] <- NA
alldata[which(is.na(alldata$Balling) == TRUE | is.na(alldata$Density) == TRUE),"Densi
ty_model_intercept_positive"] <- NA

alldata <- alldata[,setdiff(colnames(alldata),"Density")]
```

Round setpoint values as described in previous section.

```
alldata$Pressure.Setpoint[alldata$Pressure.Setpoint %in% c(45.2,46.4,46.6,46.8)] <- 4
6.0
alldata$Bowl.Setpoint[alldata$Bowl.Setpoint %in% c(122,126)] <- 120
alldata$Bowl.Setpoint[alldata$Bowl.Setpoint == 134] <- 130
```

Add the appropriate dummy variables described previously, making them NA if the main variable is NA.

```
alldata <- data.frame(alldata,
    Hyd.Pressure1_0 = rep(0,times=nrow(alldata)),
    Hyd.Pressure2_0 = rep(0,times=nrow(alldata)),
    Hyd.Pressure2_0.2 = rep(0,times=nrow(alldata)),
    Hyd.Pressure2_0_or_0.2 = rep(0,times=nrow(alldata)),
    Hyd.Pressure3_0 = rep(0,times=nrow(alldata)),
    Hyd.Pressure3_neg1.2 = rep(0,times=nrow(alldata)),
    Hyd.Pressure3_0_or_neg1.2 = rep(0,times=nrow(alldata)),
    Mnf.Flow_neg100 = rep(0,times=nrow(alldata)),
    Mnf.Flow_neg100.2 = rep(0,times=nrow(alldata)),
    Mnf.Flow_neg100_or_neg100.2 = rep(0,times=nrow(alldata)),
    Mnf.Flow_0.2 = rep(0,times=nrow(alldata)),
    Mnf.Flow_neg100_or_neg100.2_or_0.2 = rep(0,times=nrow(alldata)),
    stringsAsFactors=FALSE)
```

```
vars_to_test <- rep(c("Hyd.Pressure1","Hyd.Pressure2","Hyd.Pressure3","Mnf.Flow"),tim
es=c(1,3,3,5))
values_to_test <- vector("list",length=12)

values_to_test[c(1,2,5)] <- 0
values_to_test[c(3,11)] <- 0.2
values_to_test[[4]] <- c(0,0.2)
values_to_test[[6]] <- -1.2
values_to_test[[7]] <- c(0,-1.2)
values_to_test[[8]] <- -100
values_to_test[[9]] <- -100.2
values_to_test[[10]] <- c(-100,-100.2)
values_to_test[[12]] <- c(-100,-100.2,0.2)

names(values_to_test) <- colnames(alldata)[(ncol(alldata) - 11):ncol(alldata)]
```

```
for(i in 1:12)
{
    myvar <- vars_to_test[i]
    myval_to_test <- values_to_test[[i]]
    var_to_replace <- names(values_to_test)[i]
    ind_matching <- which(alldata[,myvar] %in% myval_to_test)
    ind_NA <- which(is.na(alldata[,myvar]) == TRUE)
    alldata[ind_matching,var_to_replace] <- 1
    alldata[ind_NA,var_to_replace] <- NA
}
```

Also, set aside indices where these variables are NA so we can check the imputed values later.

Set aside indices where pressure or bowl setpoint is NA, as may want to round the imputed values later on.

Oh, and set aside missing for brand code, as may want to assign based on probabilities of each.

```
missing_in_hyd.pressure1 <- which(is.na(alldata[,"Hyd.Pressure1"]) == TRUE)
missing_in_hyd.pressure_2and3 <- which(is.na(alldata[,"Hyd.Pressure3"]) == TRUE)
missing_mnf_flow <- which(is.na(alldata[,"Mnf.Flow"]) == TRUE)
missing_pressure_setpoint <- which(is.na(alldata$Pressure.Setpoint) == TRUE)
missing_bowl_setpoint <- which(is.na(alldata$Bowl.Setpoint) == TRUE)
missing_brand_code <- which(alldata_original$Brand.Code == "")
```

Impute missing values.

Ready to impute.

```
alldata_imputed <- impute.knn(as.matrix(alldata),k=10,rng.seed=1392)
```

```
## Cluster size 2838 broken into 284 2554
## Done cluster 284
## Cluster size 2554 broken into 594 1960
## Done cluster 594
## Cluster size 1960 broken into 710 1250
## Done cluster 710
## Done cluster 1250
## Done cluster 1960
## Done cluster 2554
```

```
alldata_imputed <- data.frame(alldata_imputed$data,check.names=FALSE,stringsAsFactors
=FALSE)
```

Look at imputed values for select variables, and round as appropriate.

```
#alldata_imputed[which(is.na(alldata$Density_model_intercept_positive) == TRUE),c("Ba
lling","Density_model_intercept_positive")]
alldata_imputed$Density_model_intercept_positive[alldata_imputed$Density_model_interc
ept_positive > 0.5] <- 1
```

```
alldata_imputed[missing_pressure_setpoint,"Pressure.Setpoint"] <- round(alldata_imput
ed[missing_pressure_setpoint,"Pressure.Setpoint"]/2)*2
alldata_imputed[missing_bowl_setpoint,"Bowl.Setpoint"] <- round(alldata_imputed[missi
ng_bowl_setpoint,"Bowl.Setpoint"]/10)*10
```

```
#alldata_imputed[missing_in_hyd.pressure1,c("Hyd.Pressure1","Hyd.Pressure1_0")]

predicted_nonzero <- missing_in_hyd.pressure1[which(alldata_imputed[missing_in_hyd.pr
essure1,"Hyd.Pressure1"] != 0)]

alldata_imputed[predicted_nonzero,"Hyd.Pressure1_0"] <- 0
```

```
#options(width=300)
#to_print <- round(alldata_imputed[missing_in_hyd.pressure_2and3,c("Hyd.Pressure2",na
mes(values_to_test)[2:4])],digits=3)
#to_print[order(to_print[,1]),]

predicted_high <- missing_in_hyd.pressure_2and3[which(alldata_imputed[missing_in_hyd.
pressure_2and3,"Hyd.Pressure2"] > 5)]

alldata_imputed[predicted_high,c("Hyd.Pressure2_0","Hyd.Pressure2_0.2","Hyd.Pressure2
_0_or_0.2")] <- 0

predicted_low <- missing_in_hyd.pressure_2and3[which(alldata_imputed[missing_in_hyd.p
ressure_2and3,"Hyd.Pressure2"] > 0 & alldata_imputed[missing_in_hyd.pressure_2and3,"H
yd.Pressure2"] < 5)]

alldata_imputed[predicted_low,"Hyd.Pressure2"] <- 0
alldata_imputed[predicted_low,c("Hyd.Pressure2_0","Hyd.Pressure2_0_or_0.2")] <- 1
```

```
#options(width=300)
#to_print <- round(alldata_imputed[missing_in_hyd.pressure_2and3,c("Hyd.Pressure3",na
mes(values_to_test)[5:7])],digits=3)
#to_print[order(to_print[,1]),]

predicted_high <- missing_in_hyd.pressure_2and3[which(alldata_imputed[missing_in_hyd.
pressure_2and3,"Hyd.Pressure3"] > 5)]

alldata_imputed[predicted_high,names(values_to_test)[5:7]] <- 0

predicted_low <- missing_in_hyd.pressure_2and3[which(alldata_imputed[missing_in_hyd.p
ressure_2and3,"Hyd.Pressure3"] > 0 & alldata_imputed[missing_in_hyd.pressure_2and3,"H
yd.Pressure3"] < 5)]

alldata_imputed[predicted_low,"Hyd.Pressure3"] <- 0
alldata_imputed[predicted_low,names(values_to_test)[c(5,7)]] <- 1
```

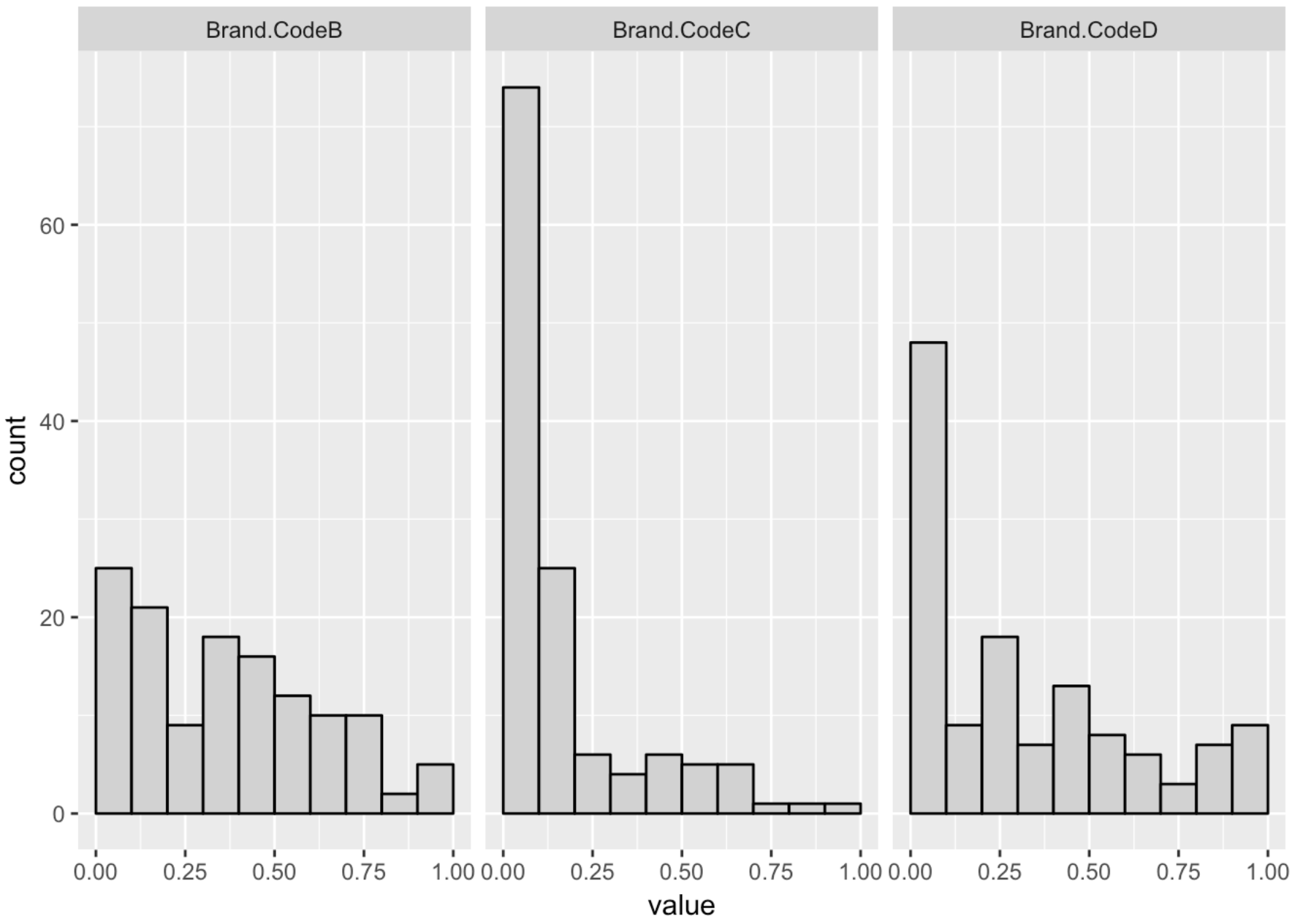
```
#options(width=300)
#to_print <- round(alldata_imputed[missing_mnf_flow,c("Mnf.Flow",names(values_to_test
)[8:12])],digits=3)
#to_print[order(to_print[,1]),]

alldata_imputed[missing_mnf_flow[1],c("Mnf.Flow",names(values_to_test)[8:12])] <- c(-
100.2,0,1,1,0,1)
alldata_imputed[missing_mnf_flow[2],c("Mnf.Flow",names(values_to_test)[8:12])] <- c(0
.2,0,0,0,1,1)
```

What does the distribution of imputed values look like for each dummy var for brand?

```
previously_missing_brand_imputed <- alldata_imputed[missing_brand_code,c("Brand.CodeB
","Brand.CodeC","Brand.CodeD")]

ggplot(gather(previously_missing_brand_imputed),
    aes(x=value)) +
    geom_histogram(breaks=seq(from=0,to=1,by=0.1),fill="lightgrey",col="black") +
    facet_wrap(~key)
```



What cutoff would we need to use for each to get similar predicted proportions of A/B/C/D here vs. in the non-missing data?

```
round(128 - (c(1368,335,679)/sum(c(328,1368,335,679)))*128)
```

```
## [1]  63 112  96
```

```
quantile(previously_missing_brand_imputed[,"Brand.CodeB"],probs=(63/128),type=4)
```

```
## 49.21875%
##     0.375
```

```
quantile(previously_missing_brand_imputed[,"Brand.CodeC"],probs=(112/128),type=4)
```

```
## 87.5%
##   0.5
```

```
quantile(previously_missing_brand_imputed[,"Brand.CodeD"],probs=(96/128),type=4)
```

```
##       75%
## 0.5555556
```

I would lean toward lowering these cutoffs a bit, as we don't want to just default to brand A too much.

If we round all values over 1/3 for Brand.CodeB to 1, 0.45 for C, and over 0.5 for D, and round down to 0 otherwise, how many observations could we disambiguate?

And how would predicted brand codes compare to previous?

```
previously_missing_brand_imputed[,1] <- ifelse(previously_missing_brand_imputed[,1] >
(1/3),1,0)
previously_missing_brand_imputed[,2] <- ifelse(previously_missing_brand_imputed[,2] >
0.45,1,0)
previously_missing_brand_imputed[,3] <- ifelse(previously_missing_brand_imputed[,3] >
0.5,1,0)


(c(328,1368,335,679)/sum(c(328,1368,335,679)))*128
```

```
## [1] 15.49225 64.61402 15.82288 32.07085
```

```
table(rowSums(previously_missing_brand_imputed))
```

```
##
##   0   1   2
##  13 113   2
```

```
apply(previously_missing_brand_imputed,2,table)
```

```
##   Brand.CodeB Brand.CodeC Brand.CodeD
## 0          61         111          95
## 1          67          17          33
```

Two observations now ambiguous, but otherwise looks good!

Let's look at the ones that are now ambiguous.

```
alldata_imputed[missing_brand_code[rowSums(previously_missing_brand_imputed) == 2],c(
"Brand.CodeB","Brand.CodeC","Brand.CodeD")]
```

```
##      Brand.CodeB Brand.CodeC Brand.CodeD
## 475          0.4         0.0         0.6
## 2023         0.4         0.6         0.0
```

Let's just assign to B in both of these instances, as that is the most common brand.

```
previously_missing_brand_imputed[which(rowSums(previously_missing_brand_imputed) == 2
)[1],"Brand.CodeD"] <- 0
previously_missing_brand_imputed[which(rowSums(previously_missing_brand_imputed) == 2
),"Brand.CodeC"] <- 0
```

```
alldata_imputed[missing_brand_code,"Brand.CodeB"] <- previously_missing_brand_imputed
[,1]
alldata_imputed[missing_brand_code,"Brand.CodeC"] <- previously_missing_brand_imputed
[,2]
alldata_imputed[missing_brand_code,"Brand.CodeD"] <- previously_missing_brand_imputed
[,3]
```

Automated pre-processing

Now on to automated pre-processing with preProcess from the caret package.

We will definitely want to center and scale the data for sure.

Then, we may or may not want to apply Yeo-Johnson (similar to Box-Cox but allows zero or negative values). These transformations may make the data more normally distributed. Which is something we would want for linear models, though may be less necessary for other models.

```
alldata_imputed_centered_scaled <- preProcess(alldata_imputed,method=c("center","scal
e"))
alldata_imputed_centered_scaled <- predict(alldata_imputed_centered_scaled,alldata_im
puted)

alldata_imputed_centered_scaled_YeoJohnson <- preProcess(alldata_imputed,method=c("ce
nter","scale","YeoJohnson"))
alldata_imputed_centered_scaled_YeoJohnson <- predict(alldata_imputed_centered_scaled
_YeoJohnson,alldata_imputed)
```

Now, let's separate training and test data.

Also, there were four observations in training set with an NA for pH. Let's remove these.

```
alldata_imputed_centered_scaled_training <- alldata_imputed_centered_scaled[training_
vs_test_vector == "Training",]
alldata_imputed_centered_scaled_test <- alldata_imputed_centered_scaled[training_vs_t
est_vector == "Test",]

alldata_imputed_centered_scaled_YeoJohnson_training <- alldata_imputed_centered_scale
d_YeoJohnson[training_vs_test_vector == "Training",]
alldata_imputed_centered_scaled_YeoJohnson_test <- alldata_imputed_centered_scaled_Ye
oJohnson[training_vs_test_vector == "Test",]
```

```
alldata_imputed_centered_scaled_training <- alldata_imputed_centered_scaled_training[
which(is.na(training_target) == FALSE),]
alldata_imputed_centered_scaled_YeoJohnson_training <- alldata_imputed_centered_scale
d_YeoJohnson_training[which(is.na(training_target) == FALSE),]

training_target_minus_NA <- training_target[which(is.na(training_target) == FALSE)]
```

Modeling

Linear model

Let's start by trying out a linear model.

Since many predictor variables are correlated, we should run partial least squares regression rather than a normal linear regression.

Try with and without Yeo-Johnson transformation.

```
pls_model_no_YeoJohnson <- plsr(pH ~ .,
    data = data.frame(pH = training_target_minus_NA,alldata_imputed_centered_scaled_t
raining,stringsAsFactors=FALSE),
    validation = "CV",segments = 10,segment.type = "random")

pls_model_with_YeoJohnson <- plsr(pH ~ .,
    data = data.frame(pH = training_target_minus_NA,alldata_imputed_centered_scaled_Y
eoJohnson_training,stringsAsFactors=FALSE),
    validation = "CV",segments = 10,segment.type = "random")
```

Print the root mean squared error of prediction (RMSEP) for the training data.

```
rmsep_no_YeoJohnson <- RMSEP(pls_model_no_YeoJohnson,newdata=data.frame(pH = training
_target_minus_NA,alldata_imputed_centered_scaled_training,stringsAsFactors=FALSE))
rmsep_with_YeoJohnson <- RMSEP(pls_model_with_YeoJohnson,newdata=data.frame(pH = trai
ning_target_minus_NA,alldata_imputed_centered_scaled_YeoJohnson_training,stringsAsFac
tors=FALSE))
```

```
min(as.numeric(as.vector(rmsep_no_YeoJohnson$val)))
```

```
## [1] 0.1291248
```

```
min(as.numeric(as.vector(rmsep_with_YeoJohnson$val)))
```

```
## [1] 0.128828
```

Looks like error is slightly decreased when run Yeo-Johnson transformation before partial least squares regression.

However, the difference is very minimal.

Will not bother with Yeo-Johnson for non-linear models, since those should be better at handling non-normal data anyway.

Create a data frame and list with not Yeo-Johnson transformed data for subsequent analysis.

```
training_centered_scaled_incl_target_dat <- data.frame(pH = training_target_minus_NA,
alldata_imputed_centered_scaled_training)
training_centered_scaled_incl_target_list <- list(x = alldata_imputed_centered_scaled
_training,y = training_target_minus_NA)
```

Nonlinear regression models

Let’s try a few different options for nonlinear regression models.

Only one I cannot run right now is MARS, as I do not have the earth package on my personal laptop.

Neural networks

Let’s try the neural networks method with model averaging.

Edit: Actually skipping this section due to long runtime. Will run it tomorrow on my work laptop when I also run the MARS model.

```
set.seed(1392)

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

averaging_nnet_model <- train(training_centered_scaled_incl_target_list$x,training_ce
ntered_scaled_incl_target_list$y,
    method="avNNet",
    tuneGrid = nnetGrid,
    trControl = trainControl(method = "cv", number = 10),
    linout = TRUE,
    trace=FALSE,
    maxit = 500)
```

Multivariate Adaptive Regression Splines (MARS)

Not actually running this right now.

```
set.seed(1392)

MARS_model <- earth(training_centered_scaled_incl_target_list$x,training_centered_sca
led_incl_target_list$y)
```

Support Vector Machine (SVM)

I tried running SVM with a polynomial kernel here, but runtime was impractically long.

However, I will try running with both a linear and radial basis function kernel.

```
set.seed(1392)

SVM_linear_model <- train(training_centered_scaled_incl_target_list$x,training_center
ed_scaled_incl_target_list$y,
    method="svmLinear",
    trControl = trainControl(method = "cv", number = 10),
    tuneLength = 14)
```

```
set.seed(1392)

SVM_RBF_model <- train(training_centered_scaled_incl_target_list$x,training_centered_
scaled_incl_target_list$y,
    method="svmRadial",
    trControl = trainControl(method = "cv", number = 10),
    tuneLength = 14)
```

K-Nearest Neighbors (KNN)

Let's run KNN just as it was run in the question example for K+J 7.2.

```
set.seed(1392)

knn_model <- train(training_centered_scaled_incl_target_list$x,training_centered_scal
ed_incl_target_list$y,
                   method = "knn",
                   tuneLength = 10)
```

Comparing nonlinear regression models

Get error and R-squared on training data for all nonlinear regression models.

```
error_and_Rsquared_train <- function(model){
    predictions <- predict(model,newdata = training_centered_scaled_incl_target_list$
x)
    return(postResample(pred = predictions, obs = training_centered_scaled_incl_targe
t_list$y))
}
```

```
for(model in c("averaging_nnet_model","MARS_model","SVM_linear_model","SVM_RBF_model"
,"knn_model"))
{
    print(model)
    print(error_and_Rsquared_train(get(model)))
}
```

```
## [1] "SVM_linear_model"
##       RMSE   Rsquared        MAE
## 0.13115766 0.42524175 0.09907155
## [1] "SVM_RBF_model"
##       RMSE   Rsquared        MAE
## 0.01673583 0.99649367 0.01649902
## [1] "knn_model"
##       RMSE   Rsquared        MAE
## 0.10757239 0.61664875 0.08111072
```

Performance of SVM with RBF kernel is quite good!

Although, I've found that SVM performance with same kernel can vary widely if you run multiple times with different seeds. So would be curious to see what others in the group have gotten, if anyone tried this model.

Regression Trees and Rule-Based Models

Let's try a few different models using regression trees or rule-based models.

Let's try random forest, cubist, and boosted trees.

Edit: Skipping randomForest and boosted trees for now. Taking too long, going to run it in the background at work.

```
set.seed(1392)

rf_model <- train(training_centered_scaled_incl_target_list$x,training_centered_scale
d_incl_target_list$y,method="rf")
```

```
set.seed(1392)

cubist_model <- train(training_centered_scaled_incl_target_list$x,training_centered_s
caled_incl_target_list$y,method="cubist")
```

```
set.seed(1392)

gbm_model <- train(training_centered_scaled_incl_target_list$x,training_centered_scal
ed_incl_target_list$y,method="gbm")
```

Regression trees and rule-based models evaluation

```
for(model in c("rf_model","cubist_model","gbm_model"))
{
    print(model)
    print(error_and_Rsquared_train(get(model)))
}
```

```
## [1] "cubist_model"
```

```
##       RMSE   Rsquared        MAE
## 0.03261859 0.96702143 0.02317437
```

Have no other models to compare to yet, but Cubist model performance seems pretty good as well. Will compare this to SVM with RBF kernel in the next section.

Model interpretation

So far, the best models tested are SVM with RBF kernel and Cubist.

Let's look at variable importance in both of these.

Mainly, I am curious if any of the dummy variables I created manually are super important to these models.

```
varImp_SVM_with_RBF_kernel <- varImp(SVM_RBF_model,scale=TRUE)

varImp_SVM_with_RBF_kernel
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 46)
##
##                                    Overall
## Mnf.Flow_neg100_or_neg100.2         100.00
## Mnf.Flow                             91.13
## Mnf.Flow_neg100_or_neg100.2_or_0.2   84.97
## Mnf.Flow_neg100.2                    74.93
## Bowl.Setpoint                        55.40
## Filler.Level                         48.00
## Usage.cont                           45.90
## Pressure.Setpoint                    42.98
## Carb.Flow                            39.68
## Brand.CodeC                          36.54
## Hyd.Pressure2_0                      31.24
## Hyd.Pressure3_0                      30.42
## Hyd.Pressure1_0                      26.75
## Hyd.Pressure3                        25.54
## Hyd.Pressure2_0_or_0.2               22.41
## Hyd.Pressure3_0_or_neg1.2            22.41
## Pressure.Vacuum                      22.18
## Fill.Pressure                        20.36
## Hyd.Pressure2                        17.74
## Oxygen.Filler                        13.31
```

```
varImp_Cubist <- varImp(cubist_model,scale=TRUE)

varImp_Cubist
```

```
## cubist variable importance
##
##   only 20 most important variables shown (out of 46)
##
##                 Overall
## Mnf.Flow         100.00
## Balling.Lvl       82.84
## Balling           79.10
## Pressure.Vacuum   79.10
## Alch.Rel          74.63
## Air.Pressurer     63.43
## Oxygen.Filler     60.45
## Hyd.Pressure3     58.21
## Bowl.Setpoint     54.48
## Temperature       54.48
## Carb.Rel          48.51
## Carb.Pressure1    48.51
## Carb.Flow         46.27
## Hyd.Pressure2     44.78
## Brand.CodeC       40.30
## Filler.Speed      39.55
## Hyd.Pressure1     36.57
## Usage.cont        33.58
## Filler.Level      30.60
## PC.Volume         23.88
```

None of the manually created special dummy variables are very important to the Cubist model.

However, it seems at least some version of these is listed as important in the SVM with RBF kernel model.