# R Code Reference

knitr::opts\_chunk$set(echo = TRUE)

library(tidyverse)

library(readr)

library(dplyr)

library(ggplot2)

library(reshape2)

library(scales)

library(Amelia)

library(caret)

data\_raw <- read\_csv("https://raw.githubusercontent.com/juliaDataScience-22/project2/main/studentDataToModel.csv")

data\_raw <- data\_raw |>

select(PH, everything())

summary(data\_raw)

na\_list <- sapply(data\_raw, function(x) sum(is.na(x)))

na\_df <- data.frame(

variable = names(na\_list),

count\_na = unlist(na\_list)) %>%

arrange(desc(count\_na))

na\_df %>%

summarize(sum(count\_na))

# top ten

na\_topten <- head(na\_df,10)

na\_topten

# bar plot

na\_plot2 <- na\_df %>%

ggplot(aes(x = reorder(variable,count\_na,decreasing = TRUE),

y = count\_na)) +

geom\_bar (stat="identity", fill = "lightblue") +

ggtitle ("Count of Missing Values by Variable") +

xlab ("Variable") +

ylab("Count of NA Values") +

theme(axis.text.x = element\_text(angle = 45, vjust = 1,

size = 10, hjust = 1))

na\_plot2

ggsave("missing.png", plot = na\_plot2, width = 8, height = 5)

# densityt plot from amelia package

missmap(data\_raw, x.cex = 0.7, gap.xaxis = 0,

main = "Missing Data: Density Map by Variable")

# Categorical variable

brand\_plot <-data\_raw %>%

ggplot(aes(x=`Brand Code`)) +

geom\_bar(fill = "cornsilk3") +

ggtitle("Distribution by Brand Code") +

scale\_y\_continuous(breaks = pretty\_breaks(10))

brand\_plot

ggsave("brand\_histogram.png", plot = brand\_plot, width = 8, height = 4)

# Numeric variables

data\_melt <- melt(data\_raw)

hist\_plot <- data\_melt %>%

ggplot(aes(x = value)) +

geom\_histogram(fill = "seagreen") +

facet\_wrap(~variable, scales='free\_x') +

theme(

title = element\_text(size = 14, face = "bold"),

strip.text = element\_text(size =12, face = "bold"),

axis.title = element\_text(size = 12)) +

ggtitle("Frequency by Numeric Variable: All Brand Codes")

hist\_plot

ggsave("histogram.png", plot = hist\_plot, width = 16, height = 12)

cor\_vars <- select(data\_raw, -`Brand Code`)

cor\_matrix <- cor(cor\_vars, use ="pairwise.complete.obs")

plot\_matrix <- melt(cor\_matrix)

# extract most highly correlated pairs (positive or negative)

highly\_cor <- filter(plot\_matrix, (value !=1 & (value >.6 | value < -.6))) %>% arrange(desc(value)) %>%

distinct(value, .keep\_all = TRUE)

highly\_cor

# plot all

plot\_correlations <- plot\_matrix %>%

ggplot(aes(x=Var1, y=Var2, fill=value)) +

geom\_tile() +

scale\_fill\_gradient2(low = "blue", high = "red", mid = "white",

midpoint = 0, limit = c(-1,1), space = "Lab",

name="Correlation") +

theme(axis.text.x = element\_text(angle = 45, vjust = 1,

size = 14, hjust = 1),

axis.text.y = element\_text(size = 14))

plot\_correlations

ggsave("correlations.png", plot = plot\_correlations, width = 12, height = 8)

# Box plots for detecting outliers in numerical variables

plot\_outlier <- data\_melt %>%

ggplot(aes(y = value)) +

geom\_boxplot() +

facet\_wrap(~variable, scales = 'free\_y') +

ggtitle("Box Plots for Outliers Detection")

plot\_outlier

ggsave("outliers.png", plot = plot\_outlier, width = 16, height = 12)

# numeric predictors only

numeric\_predictors <- names(data\_raw)[sapply(data\_raw, is.numeric) &

names(data\_raw) != "PH"]

# outlier function

is\_outlier <- function(x) {

return(abs(x - mean(x, na.rm = TRUE)) > 2 \* sd(x, na.rm = TRUE))

}

# loop

for (var2 in numeric\_predictors) {

var2\_fixed <- paste0("`", var2, "`")

data\_raw$outlier <- is\_outlier(data\_raw[[var2]])

scatter1 <- ggplot(data\_raw, aes\_string(x = var2\_fixed, y = "`PH`")) +

geom\_point() +

geom\_point(data = subset(data\_raw, outlier == TRUE), aes\_string(x = var2\_fixed, y = "`PH`"), color = 'red') +

labs(title = paste("Scatter Plot of", var2, "vs. PH with Outliers Highlighted"),

x = var2,

y = "PH") +

theme(title = element\_text(size = 16, face = "bold"),

axis.text.x = element\_text(size = 14),

axis.text.y = element\_text(size = 14))

print(scatter1)

ggsave(paste0("scatter-",var2\_fixed,".png"), plot = scatter1, width = 8, height = 6)

}

data\_raw <- data\_raw |>

select(-outlier)

columnTypes <- read\_csv("https://raw.githubusercontent.com/juliaDataScience-22/project2/main/dataColumnTypes.csv")

View(columnTypes)

dataToPredict <- read\_csv("https://raw.githubusercontent.com/juliaDataScience-22/project2/main/studentEvaluationToPredict.csv")

View(dataToPredict)

# Install and load the package

library(mice)

# Rename columns

library(stringr)

colnames(data\_raw) <- c(

"pH", "brandCode", "carbVolume", "fillOunces", "pcVolume", "carbPressure",

"carbTemp", "psc", "pscFill", "pscCO2", "mnfFlow", "carbPressure1",

"fillPressure", "hydPressure1", "hydPressure2", "hydPressure3", "hydPressure4",

"fillerLevel", "fillerSpeed", "temperature", "usageCont", "carbFlow", "density",

"mfr", "balling", "pressureVacuum", "oxygenFiller", "bowlSetpoint", "pressureSetpoint",

"airPressurer", "alchRel", "carbRel", "ballingLvl"

)

if (!is.factor(data\_raw$brandCode)) {

data\_raw$brandCode <- as.factor(data\_raw$brandCode)

}

methods <- c(

"pmm", "polyreg", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm"

)

# Perform multiple imputation

imputed\_data <- mice(data\_raw, m = 5, method = methods, maxit = 5, seed = 500)

# Complete the imputed data

complete\_data <- complete(imputed\_data, 1)

# Handle categorical/binary variables

encoded\_data <- model.matrix(~ . - 1, data = complete\_data)

df\_Final\_Data <- as.data.frame(encoded\_data)

# Install and load the caret package

library(caret)

# Example: Split data into 70% training and 30% testing

index <- createDataPartition(df\_Final\_Data$pH, p = 0.7, list = FALSE)

training\_data <- df\_Final\_Data[index, ]

testing\_data <- df\_Final\_Data[-index, ]

# View the dimensions of training and testing data

dim(training\_data)

dim(testing\_data)

# Drop brandCodeA to avoid perfect multicollinearity

ols\_train <- training\_data %>%

select(-brandCodeA)

# OLS Regression with stepwise selection

model <- lm(pH ~ ., data = ols\_train)

#summary(model)

# Stepwise selection

step\_model <- step(model, direction = "both", trace = 0)

summary(step\_model)

# Rewrite using the train function from the caret package to make it easier to compare with other models

# Extract the formula from the step model

stepwise\_formula <- formula(step\_model)

step\_model\_train <- train(stepwise\_formula,

data = ols\_train,

method = "lm",

trControl = caret::trainControl(method = "cv",

number = 10))

# Install and load the pls package

library(pls)

# PLS Regression with cross-validation

pls\_model\_full <- plsr(pH ~ ., data = training\_data, scale = TRUE, validation = "CV")

summary(pls\_model\_full)

# Select the top 15 components based on diminishing returns in explained variance

pls\_model1 <- plsr(pH ~ ., data = training\_data, scale = TRUE, validation = "CV", ncomp = 15)

#summary(pls\_model)

# Rewrite using the train function from the caret package to make it easier to compare with other models

tune\_grid <- expand.grid(ncomp = 1:15)

pls\_model <- train(pH ~ ., data = training\_data, method = "pls", trControl = trainControl(method = "cv", number = 10), tune\_grid = tune\_grid)

# Load necessary libraries

library(caret)

library(elasticnet)

# Elastic Net Regression with cross-validation

enet\_model\_full <- train(pH ~ ., data = training\_data, method = "enet", trControl = trainControl(method = "cv", number = 10))

# Get R-squared value

enet\_results <- enet\_model\_full$results

r\_squared <- enet\_results[enet\_results$lambda == enet\_model\_full$bestTune$lambda & enet\_results$fraction == enet\_model\_full$bestTune$fraction, "Rsquared"]

r\_squared

# Random Forest with cross-validation

rf\_model <- train(pH ~ ., data = training\_data, method = "rf", trControl = trainControl(method = "cv", number = 10))

# Cubist with cross-validation

cubist\_model <- train(pH ~ ., data = training\_data, method = "cubist", trControl = trainControl(method = "cv", number = 10))

# SVM with cross-validation

svm\_model <- train(pH ~ ., data = training\_data, method = "svmRadial", trControl = trainControl(method = "cv", number = 10))

summary(svm\_model)

# Plot the results

plot(svm\_model)

# Compare the models

resamples <- resamples(list(OLS = step\_model\_train, PLS = pls\_model, ElasticNet = enet\_model\_full, Random\_Forest = rf\_model, Cubist = cubist\_model, SVM = svm\_model))

resample\_summary <- summary(resamples)

library(knitr)

# Convert summary to data frame

MAE\_df <- as.data.frame(resample\_summary$statistics$MAE)

MAE\_df$Metric <- "MAE"

MAE\_df$Model <- rownames(MAE\_df)

rownames(MAE\_df) <- NULL

RMSE\_df <- as.data.frame(resample\_summary$statistics$RMSE)

RMSE\_df$Metric <- "RMSE"

RMSE\_df$Model <- rownames(RMSE\_df)

rownames(RMSE\_df) <- NULL

Rsquared\_df <- as.data.frame(resample\_summary$statistics$Rsquared)

Rsquared\_df$Metric <- "Rsquared"

Rsquared\_df$Model <- rownames(Rsquared\_df)

rownames(Rsquared\_df) <- NULL

# Combine all data frames

combined\_df <- bind\_rows(MAE\_df, RMSE\_df, Rsquared\_df)

# Melt the data frame for easier manipulation

melted\_df <- melt(combined\_df, id.vars = c("Model", "Metric"))

# Reorder the columns for better readability

final\_df <- dcast(melted\_df, Model + Metric ~ variable)

# Display the final dataframe

kable(final\_df)

# Tune the Cubist model

tune\_grid <- expand.grid(committees = seq(10, 100, by = 10), neighbors = c(0,3,5,7,9))

tuned\_cubist\_model <- train(pH ~ ., data = training\_data, method = "cubist", trControl = trainControl(method = "cv", number = 10), tuneGrid = tune\_grid)

# Evaluate all the models on the testing data

ols\_pred <- predict(step\_model, newdata = testing\_data)

pls\_pred <- predict(pls\_model, newdata = testing\_data)

enet\_pred <- predict(enet\_model\_full, newdata = testing\_data)

rf\_pred <- predict(rf\_model, newdata = testing\_data)

cubist\_first\_pred <- predict(cubist\_model, newdata = testing\_data)

svm\_pred <- predict(svm\_model, newdata = testing\_data)

cubist\_tuned\_pred <- predict(tuned\_cubist\_model, newdata = testing\_data)

# Calculate the MAPE for each model

mape\_ols <- mean(abs(testing\_data$pH - ols\_pred) / testing\_data$pH)

mape\_pls <- mean(abs(testing\_data$pH - pls\_pred) / testing\_data$pH)

mape\_enet <- mean(abs(testing\_data$pH - enet\_pred) / testing\_data$pH)

mape\_rf <- mean(abs(testing\_data$pH - rf\_pred) / testing\_data$pH)

mape\_cubist\_first <- mean(abs(testing\_data$pH - cubist\_first\_pred) / testing\_data$pH)

mape\_svm <- mean(abs(testing\_data$pH - svm\_pred) / testing\_data$pH)

mape\_cubist\_tuned <- mean(abs(testing\_data$pH - cubist\_tuned\_pred) / testing\_data$pH)

# Calculate the RMSE for each model

rmse\_ols <- sqrt(mean((testing\_data$pH - ols\_pred)^2))

rmse\_pls <- sqrt(mean((testing\_data$pH - pls\_pred)^2))

rmse\_enet <- sqrt(mean((testing\_data$pH - enet\_pred)^2))

rmse\_rf <- sqrt(mean((testing\_data$pH - rf\_pred)^2))

rmse\_cubist\_first <- sqrt(mean((testing\_data$pH - cubist\_first\_pred)^2))

rmse\_svm <- sqrt(mean((testing\_data$pH - svm\_pred)^2))

rmse\_cubist\_tuned <- sqrt(mean((testing\_data$pH - cubist\_tuned\_pred)^2))

# Create a data frame with the evaluation metrics

evaluation\_metrics <- data.frame(

Model = c("OLS", "PLS", "ElasticNet", "Random\_Forest", "Cubist\_First", "SVM", "Cubist\_Tuned"),

MAPE = c(mape\_ols, mape\_pls, mape\_enet, mape\_rf, mape\_cubist\_first, mape\_svm, mape\_cubist\_tuned),

RMSE = c(rmse\_ols, rmse\_pls, rmse\_enet, rmse\_rf, rmse\_cubist\_first, rmse\_svm, rmse\_cubist\_tuned)

)

kable(evaluation\_metrics)

# Extract variable importance from the Cubist model

var\_importance <- varImp(tuned\_cubist\_model)

# Plot variable importance

plot(var\_importance)

# prp plots

library(pdp)

# Extract the partial dependence of the most important variables

pdp\_mnfFlow <- partial(tuned\_cubist\_model, pred.var = "mnfFlow")

pdp\_ballingLvl <- partial(tuned\_cubist\_model, pred.var = "ballingLvl")

pdp\_balling <- partial(tuned\_cubist\_model, pred.var = "balling")

pdp\_alchRel <- partial(tuned\_cubist\_model, pred.var = "alchRel")

pdp\_pressureVacuum <- partial(tuned\_cubist\_model, pred.var = "pressureVacuum")

# Plot the partial dependence

plot(pdp\_mnfFlow)

plot(pdp\_ballingLvl)

plot(pdp\_balling)

plot(pdp\_alchRel)

plot(pdp\_pressureVacuum)

# Do all the same data prep steps as before

# Drop the ph column

dataToPredict <- dataToPredict |>

select(-PH)

# Rename columns

colnames(dataToPredict) <- c(

"brandCode", "carbVolume", "fillOunces", "pcVolume", "carbPressure",

"carbTemp", "psc", "pscFill", "pscCO2", "mnfFlow", "carbPressure1",

"fillPressure", "hydPressure1", "hydPressure2", "hydPressure3", "hydPressure4",

"fillerLevel", "fillerSpeed", "temperature", "usageCont", "carbFlow", "density",

"mfr", "balling", "pressureVacuum", "oxygenFiller", "bowlSetpoint", "pressureSetpoint",

"airPressurer", "alchRel", "carbRel", "ballingLvl"

)

if (!is.factor(dataToPredict$brandCode)) {

dataToPredict$brandCode <- as.factor(dataToPredict$brandCode)

}

methods <- c(

"pmm", "polyreg", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm", "pmm",

"pmm", "pmm", "pmm", "pmm"

)

# Perform multiple imputation

imputed\_eval <- mice(dataToPredict, m = 5, method = 'pmm', maxit = 5, seed = 500)

# Complete the imputed data

complete\_eval <- complete(imputed\_eval, 1)

# Handle categorical/binary variables

encoded\_eval <- model.matrix(~ . - 1, data = complete\_eval)

df\_Final\_Eval <- as.data.frame(encoded\_eval)

# Check for NAs

sum(is.na(complete\_eval$brandCode))

# find missing rows in df\_Final\_Eval

colnames(df\_Final\_Eval)

# Predict the pH value

predictions <- predict(tuned\_cubist\_model, newdata = df\_Final\_Eval)

# Add the predictions to the dataToPredict dataset

dataToPredict$pH <- predictions

# View the dataToPredict dataset with the predicted pH values

View(dataToPredict)

write.csv(dataToPredict, 'export\_data\_for\_submission.csv')