Data 624: Predictive Analytics HW:2

Group 2: Alice Friedman, Kayleah Griffen, Michael Ippolito, Josh Iden

# Introduction

The python work below accompanies the first two problems from Data 624, Summer 2023, HW2. The goal was to compare the available data science packages in R to those in python. For the R submission, we referenced the *Applied Predictive Modeling* textbook for what libraries to use and coding examples. For the python version, we used the anaconda python distribution which has several data science libraries pre-downloaded. As a part of this the sklearn library (https://scikit-learn.org/stable/) was particularly useful. To actually develop and compare the R version to the python versions, Rstudio was used as you can code in both R and python in Rstudio. The R library “reticulate” lets you run the python code in R and allows for you to bring in variables from R into python. One caveat of the python chunks is you have to keep all of the python code that shares python libraries and variables in the same chunk, otherwise when knitting the libraries/variables are not shared and the knit will fail due to unloaded libraries/undeclared variables.

# 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 the 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.**

### (a)

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

First the data, “ChemicalManufacturingProcess” was loaded in R, and then brought into python using the access method of putting an “r.” in front of the variable name to get an R variable into python, so in the python chunk the data was “r.ChemicalManufacturingProcess”.

r.ChemicalManufacturingProcess.shape

## (176, 58)

The data is the same shape in R and python. We can tell it has been brought into python correctly.

### (b)

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

First all of the relevant python packages are loaded that will be used throughout this section. For this imputation part the “from sklearn.impute import SimpleImputer” brings in a “SimpleImputer” from sklearn.impute. First we make a list of all of the missing values from each variable, then we use the simple imputer to impute, then we show that there are no more missing values. In R, we used KNN-imputation. In this case the base anaconda distribution does not have a KNN-imputer so the simple imputer used mean imputation. This is a shortcoming of the sklearn loaded library in anaconda.

import pandas as pd  
import numpy as np  
from numpy import arange  
  
from scipy.stats import skew  
  
import matplotlib.pyplot as plt  
from matplotlib import colors  
from matplotlib.ticker import PercentFormatter  
  
from sklearn.impute import SimpleImputer  
from sklearn.feature\_selection import VarianceThreshold  
from sklearn.preprocessing import RobustScaler  
from sklearn.preprocessing import PowerTransformer  
from sklearn.model\_selection import train\_test\_split  
from sklearn import model\_selection  
from sklearn.model\_selection import RepeatedKFold  
from sklearn.cross\_decomposition import PLSRegression  
from sklearn.metrics import mean\_squared\_error  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import RepeatedKFold  
from sklearn.linear\_model import ElasticNetCV

r.ChemicalManufacturingProcess.isna().sum()

## Yield 0  
## BiologicalMaterial01 0  
## BiologicalMaterial02 0  
## BiologicalMaterial03 0  
## BiologicalMaterial04 0  
## BiologicalMaterial05 0  
## BiologicalMaterial06 0  
## BiologicalMaterial07 0  
## BiologicalMaterial08 0  
## BiologicalMaterial09 0  
## BiologicalMaterial10 0  
## BiologicalMaterial11 0  
## BiologicalMaterial12 0  
## ManufacturingProcess01 1  
## ManufacturingProcess02 3  
## ManufacturingProcess03 15  
## ManufacturingProcess04 1  
## ManufacturingProcess05 1  
## ManufacturingProcess06 2  
## ManufacturingProcess07 1  
## ManufacturingProcess08 1  
## ManufacturingProcess09 0  
## ManufacturingProcess10 9  
## ManufacturingProcess11 10  
## ManufacturingProcess12 1  
## ManufacturingProcess13 0  
## ManufacturingProcess14 1  
## ManufacturingProcess15 0  
## ManufacturingProcess16 0  
## ManufacturingProcess17 0  
## ManufacturingProcess18 0  
## ManufacturingProcess19 0  
## ManufacturingProcess20 0  
## ManufacturingProcess21 0  
## ManufacturingProcess22 1  
## ManufacturingProcess23 1  
## ManufacturingProcess24 1  
## ManufacturingProcess25 5  
## ManufacturingProcess26 5  
## ManufacturingProcess27 5  
## ManufacturingProcess28 5  
## ManufacturingProcess29 5  
## ManufacturingProcess30 5  
## ManufacturingProcess31 5  
## ManufacturingProcess32 0  
## ManufacturingProcess33 5  
## ManufacturingProcess34 5  
## ManufacturingProcess35 5  
## ManufacturingProcess36 5  
## ManufacturingProcess37 0  
## ManufacturingProcess38 0  
## ManufacturingProcess39 0  
## ManufacturingProcess40 1  
## ManufacturingProcess41 1  
## ManufacturingProcess42 0  
## ManufacturingProcess43 0  
## ManufacturingProcess44 0  
## ManufacturingProcess45 0  
## dtype: int64

# define the imputer  
imputer = SimpleImputer(strategy='mean')  
  
# define the pandas dataframe to be imputed  
pdfimp = r.ChemicalManufacturingProcess  
  
# perform imputation  
imputer = imputer.fit(pdfimp)  
pdfimp.iloc[:,:] = imputer.transform(r.ChemicalManufacturingProcess)  
  
# check nas and shape  
pdfimp.isna().sum()

## Yield 0  
## BiologicalMaterial01 0  
## BiologicalMaterial02 0  
## BiologicalMaterial03 0  
## BiologicalMaterial04 0  
## BiologicalMaterial05 0  
## BiologicalMaterial06 0  
## BiologicalMaterial07 0  
## BiologicalMaterial08 0  
## BiologicalMaterial09 0  
## BiologicalMaterial10 0  
## BiologicalMaterial11 0  
## BiologicalMaterial12 0  
## ManufacturingProcess01 0  
## ManufacturingProcess02 0  
## ManufacturingProcess03 0  
## ManufacturingProcess04 0  
## ManufacturingProcess05 0  
## ManufacturingProcess06 0  
## ManufacturingProcess07 0  
## ManufacturingProcess08 0  
## ManufacturingProcess09 0  
## ManufacturingProcess10 0  
## ManufacturingProcess11 0  
## ManufacturingProcess12 0  
## ManufacturingProcess13 0  
## ManufacturingProcess14 0  
## ManufacturingProcess15 0  
## ManufacturingProcess16 0  
## ManufacturingProcess17 0  
## ManufacturingProcess18 0  
## ManufacturingProcess19 0  
## ManufacturingProcess20 0  
## ManufacturingProcess21 0  
## ManufacturingProcess22 0  
## ManufacturingProcess23 0  
## ManufacturingProcess24 0  
## ManufacturingProcess25 0  
## ManufacturingProcess26 0  
## ManufacturingProcess27 0  
## ManufacturingProcess28 0  
## ManufacturingProcess29 0  
## ManufacturingProcess30 0  
## ManufacturingProcess31 0  
## ManufacturingProcess32 0  
## ManufacturingProcess33 0  
## ManufacturingProcess34 0  
## ManufacturingProcess35 0  
## ManufacturingProcess36 0  
## ManufacturingProcess37 0  
## ManufacturingProcess38 0  
## ManufacturingProcess39 0  
## ManufacturingProcess40 0  
## ManufacturingProcess41 0  
## ManufacturingProcess42 0  
## ManufacturingProcess43 0  
## ManufacturingProcess44 0  
## ManufacturingProcess45 0  
## dtype: int64

pdfimp.shape

## (176, 58)

### (c)

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

Next several pre-processing steps were taken. First near zero variance variables were detected and removed, the python library lets you specify a threshold for NZV which we chose at 0.02. This resulted in the removal of 6 variables. Next we checked for outliers. We rewrote the same function from R into python to print out how many outliers there were for each predictor. Next we examined skewness and printed skewness greater than 2 or less than -2. Due to the presence of outliers, we used the “Robust Scaler” because it centers and scales in a way that doesn’t allow the outliers to have a big impact on the center/scale values. Then to address skewness we used the “Yeo-Johnson” transformation. We also used a correlation plot to get an understanding for highly correlated variables, and upon seeing highly correlated variables opted to use a modeling technique that was robust to correlated variables. Next we declared 10-fold cross validation, split the test and training data, and trained a PLS and elastic net model. The tuning for the PLS model resulted in 5 components being the best number, which is the same as what we found in R.

var\_thr = VarianceThreshold(threshold = 0.02) #Removing both constant and quasi-constant  
var\_thr.fit(pdfimp)

## VarianceThreshold(threshold=0.02)

concol = [column for column in pdfimp.columns   
 if column not in pdfimp.columns[var\_thr.get\_support()]]  
  
for features in concol:  
 print(features)

## BiologicalMaterial07  
## ManufacturingProcess03  
## ManufacturingProcess34  
## ManufacturingProcess36  
## ManufacturingProcess40  
## ManufacturingProcess41

pdf0var = pdfimp.drop(concol,axis=1)  
  
  
for i in range(1,pdf0var.shape[1]):  
 # IQR  
 # Calculate the upper and lower limits  
 Q1 = pdf0var.iloc[:, i].quantile(0.25)  
 Q3 = pdf0var.iloc[:, i].quantile(0.75)  
 IQR = Q3 - Q1  
 lower = Q1 - 1.5\*IQR  
 upper = Q3 + 1.5\*IQR  
   
 # Create arrays of Boolean values indicating the outlier rows  
 upper\_array = np.where(pdf0var.iloc[:, i]>upper)[0]  
 lower\_array = np.where(pdf0var.iloc[:, i]<lower)[0]  
   
 total = upper\_array.shape[0] + lower\_array.shape[0]  
   
 print(pdf0var.columns[i], total)

## BiologicalMaterial01 4  
## BiologicalMaterial02 0  
## BiologicalMaterial03 0  
## BiologicalMaterial04 3  
## BiologicalMaterial05 1  
## BiologicalMaterial06 1  
## BiologicalMaterial08 1  
## BiologicalMaterial09 3  
## BiologicalMaterial10 11  
## BiologicalMaterial11 6  
## BiologicalMaterial12 0  
## ManufacturingProcess01 4  
## ManufacturingProcess02 35  
## ManufacturingProcess04 2  
## ManufacturingProcess05 10  
## ManufacturingProcess06 6  
## ManufacturingProcess07 0  
## ManufacturingProcess08 0  
## ManufacturingProcess09 5  
## ManufacturingProcess10 8  
## ManufacturingProcess11 7  
## ManufacturingProcess12 34  
## ManufacturingProcess13 4  
## ManufacturingProcess14 10  
## ManufacturingProcess15 20  
## ManufacturingProcess16 14  
## ManufacturingProcess17 4  
## ManufacturingProcess18 8  
## ManufacturingProcess19 5  
## ManufacturingProcess20 8  
## ManufacturingProcess21 17  
## ManufacturingProcess22 0  
## ManufacturingProcess23 0  
## ManufacturingProcess24 0  
## ManufacturingProcess25 7  
## ManufacturingProcess26 9  
## ManufacturingProcess27 14  
## ManufacturingProcess28 0  
## ManufacturingProcess29 11  
## ManufacturingProcess30 2  
## ManufacturingProcess31 5  
## ManufacturingProcess32 3  
## ManufacturingProcess33 4  
## ManufacturingProcess35 10  
## ManufacturingProcess37 2  
## ManufacturingProcess38 5  
## ManufacturingProcess39 9  
## ManufacturingProcess42 11  
## ManufacturingProcess43 10  
## ManufacturingProcess44 9  
## ManufacturingProcess45 17

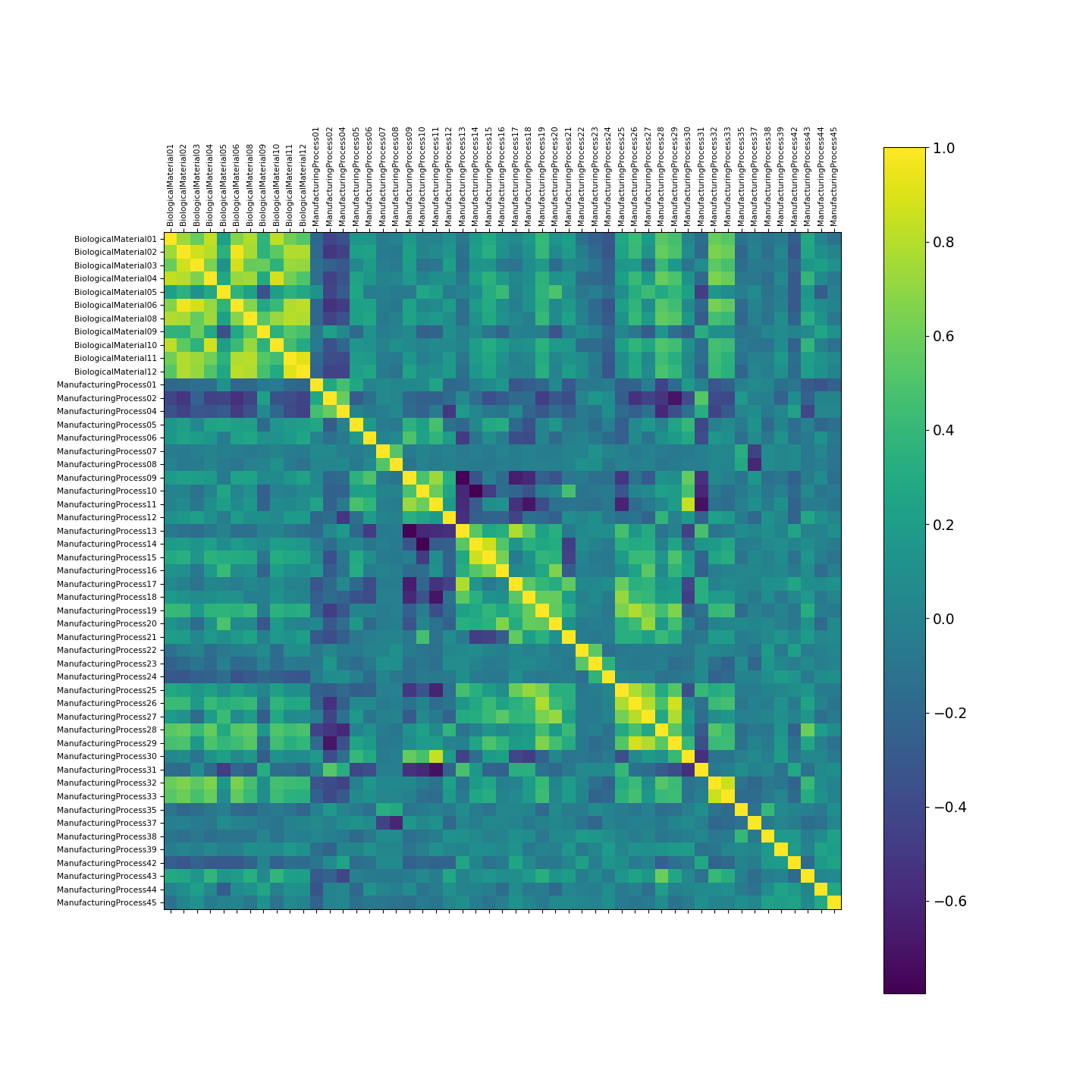
all\_skewed = []  
  
for i in range(1,pdf0var.shape[1]):  
 # Calculate the skewness  
 s = skew(pdf0var.iloc[:, i], axis=0, bias=True)  
 if (s > 2.0 or s < -2.0):  
 print(pdf0var.columns[i], s)  
 all\_skewed.append(i)

## BiologicalMaterial10 2.422999527598718  
## ManufacturingProcess01 -3.9653098909818074  
## ManufacturingProcess05 2.6170584191099233  
## ManufacturingProcess06 3.0858972050204163  
## ManufacturingProcess16 -12.526835771049369  
## ManufacturingProcess18 -12.845460479160392  
## ManufacturingProcess20 -12.74680990314572  
## ManufacturingProcess25 -12.927589490141536  
## ManufacturingProcess26 -12.966909319094245  
## ManufacturingProcess27 -12.811379385877506  
## ManufacturingProcess29 -10.32159764913633  
## ManufacturingProcess30 -4.86738793204111  
## ManufacturingProcess31 -12.100698922010961  
## ManufacturingProcess39 -4.305766123192013  
## ManufacturingProcess42 -5.496789259937433  
## ManufacturingProcess43 9.132598657923342  
## ManufacturingProcess44 -5.013019018021355  
## ManufacturingProcess45 -4.112944720676876

pdfx = pdf0var.iloc[:,1:]  
  
  
  
  
robust\_scaler = RobustScaler()  
pdfscaled = robust\_scaler.fit\_transform(pdfx.to\_numpy())  
pdfscaled = pd.DataFrame(pdfscaled, columns=pdfx.columns)  
  
  
# can run this to see the plots  
# for i in range(1,pdfx.shape[1]):  
# fig, axs = plt.subplots(1, 2, sharey=True, tight\_layout=True)  
# n\_bins = 30  
# axs[0].hist(pdfx.iloc[:, i], bins=n\_bins)  
# axs[1].hist(pdfscaled.iloc[:, i], bins=n\_bins)  
# plt.show()  
   
  
power\_transfromer = PowerTransformer(method='yeo-johnson', standardize=False)  
pdfyeo = power\_transfromer.fit\_transform(pdfscaled.to\_numpy())  
pdfyeo = pd.DataFrame(pdfyeo, columns=pdfscaled.columns)  
  
# for i in range(1,pdfscaled.shape[1]):  
# fig, axs = plt.subplots(1, 2, sharey=True, tight\_layout=True)  
# n\_bins = 30  
# axs[0].hist(pdfscaled.iloc[:, i], bins=n\_bins)  
# axs[1].hist(pdfyeo.iloc[:, i], bins=n\_bins)  
# plt.show()  
  
  
f = plt.figure(figsize=(15, 15))  
plt.matshow(pdfyeo.corr(), fignum=f.number)  
plt.xticks(range(pdfyeo.select\_dtypes(['number']).shape[1]), pdfyeo.select\_dtypes(['number']).columns, fontsize=8, rotation=90)

plt.yticks(range(pdfyeo.select\_dtypes(['number']).shape[1]), pdfyeo.select\_dtypes(['number']).columns, fontsize=8)

cb = plt.colorbar()  
cb.ax.tick\_params(labelsize=14)  
plt.show()



x = pdfyeo  
y = r.ChemicalManufacturingProcess["Yield"]  
   
# using the train test split function  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y  
,random\_state=104,  
test\_size=0.2,   
shuffle=True)  
  
  
#define cross-validation method  
cv = RepeatedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
  
mse = []  
n = len(x)  
  
# Calculate MSE with only the intercept  
score = -1\*model\_selection.cross\_val\_score(PLSRegression(n\_components=1),  
 np.ones((n,1)), y, cv=cv, scoring='neg\_mean\_squared\_error').mean()

mse.append(score)  
  
# Calculate MSE using cross-validation, adding one component at a time  
for i in range(1, 10):  
 pls = PLSRegression(n\_components=i)  
 score = -1\*model\_selection.cross\_val\_score(pls, x, y, cv=cv,  
 scoring='neg\_mean\_squared\_error').mean()  
 mse.append(score)  
   
#plot test MSE vs. number of components  
plt.clf()  
plt.plot(mse)  
plt.xlabel('Number of PLS Components')  
plt.ylabel('MSE')  
plt.show()

#calculate RMSE  
pls = PLSRegression(n\_components=5)  
pls.fit((x\_train), y\_train)

## PLSRegression(n\_components=5)

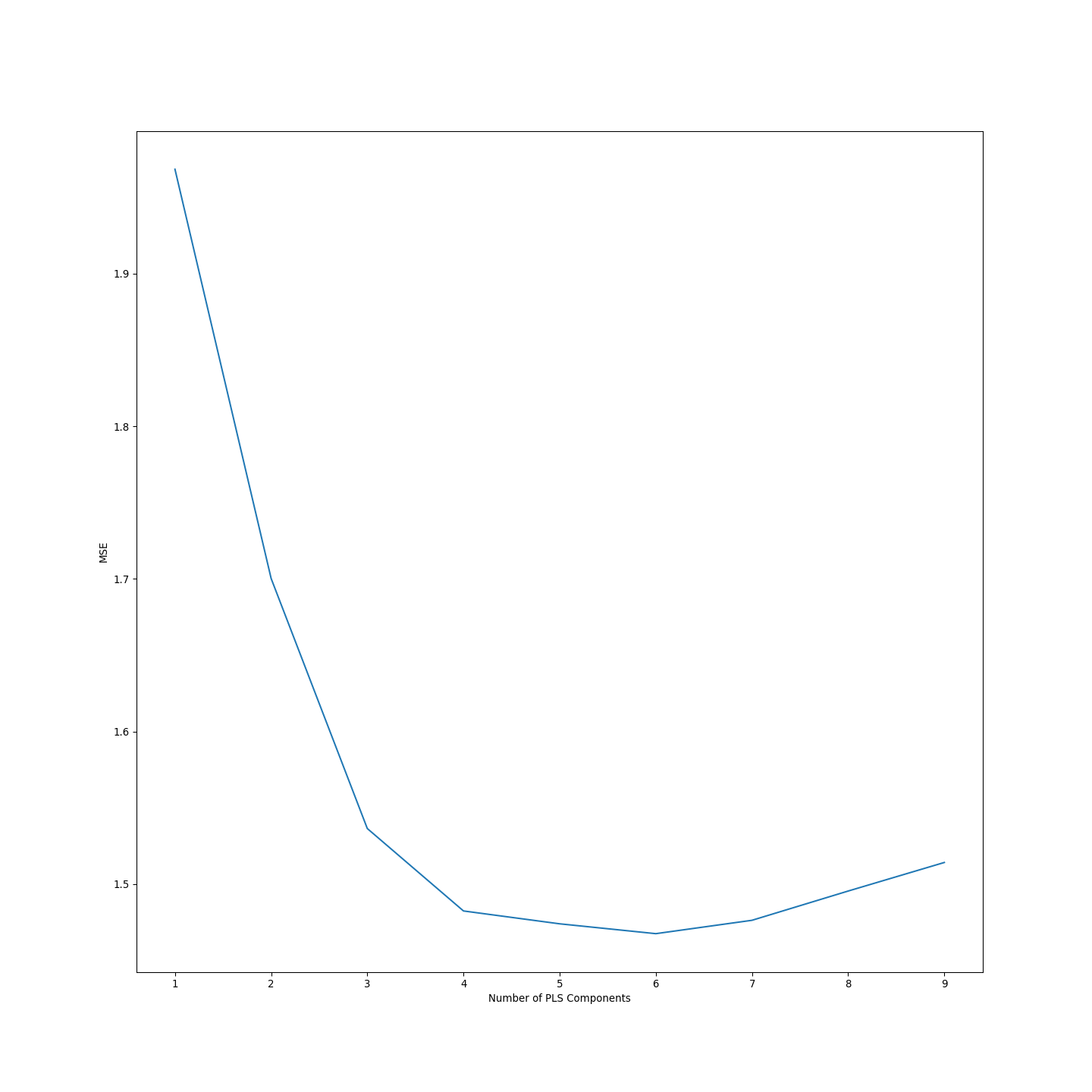
np.sqrt(mse[6])

## 1.2114255280709894

alphas = [0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 1]  
  
elastic\_cv=ElasticNetCV(alphas=alphas, cv=cv)  
enet = elastic\_cv.fit(x\_train, y\_train)

print(enet.alpha\_)

## 0.1



### (d)

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

Next we checked the RMSE for the PLS and ENET models. The RMSE of the PLS for the test data was 1.07, in comparison to 1.211 for the training – so it actually behaved slightly better. We did not find the RMSE for the training of the enet, but on the test it was 1.125 – slightly worse than the PLS model.

np.sqrt(mean\_squared\_error(y\_test, pls.predict((x\_test))))

## 1.0744687386335727

np.sqrt(mean\_squared\_error(y\_test, enet.predict(x\_test)))

## 1.1252171455988855

### (e)

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

Through the available anaconda libraries, we did not find a function to call for the variable importance, for that reason we looked online for how to calculate this and found: <https://www.researchgate.net/post/How-can-I-compute-Variable-Importance-in-Projection-VIP-in-Partial-Least-Squares-PLS> . We used the function from this forum to find the VIPs in python. Our top variable was the same, along with most of the top 10 from R to python. Manufacturing processes dominated the variable importance.

def \_calculate\_vips(model):  
 t = model.x\_scores\_  
 w = model.x\_weights\_  
 q = model.y\_loadings\_  
 p, h = w.shape  
 vips = np.zeros((p,))  
 s = np.diag(np.matmul(np.matmul(np.matmul(t.T,t),q.T), q)).reshape(h, -1)  
 total\_s = np.sum(s)  
 for i in arange(p):  
 weight = np.array([ (w[i,j] / np.linalg.norm(w[:,j]))\*\*2 for j in range(h) ])  
 vips[i] = np.sqrt(p\*(np.matmul(s.T, weight))/total\_s)  
 return vips  
  
vimps = \_calculate\_vips(pls)  
  
vimpsdf = pd.DataFrame({'Var' : pdfscaled.columns, 'VarImp' : vimps})  
  
vimpsdf.sort\_values(by = "VarImp", ascending = False)

## Var VarImp  
## 41 ManufacturingProcess32 2.044309  
## 22 ManufacturingProcess13 1.699199  
## 18 ManufacturingProcess09 1.602227  
## 26 ManufacturingProcess17 1.560868  
## 5 BiologicalMaterial06 1.424551  
## 1 BiologicalMaterial02 1.409760  
## 2 BiologicalMaterial03 1.330042  
## 42 ManufacturingProcess33 1.298524  
## 6 BiologicalMaterial08 1.276271  
## 15 ManufacturingProcess06 1.241981  
## 3 BiologicalMaterial04 1.235088  
## 21 ManufacturingProcess12 1.219019  
## 10 BiologicalMaterial12 1.213714  
## 9 BiologicalMaterial11 1.187432  
## 20 ManufacturingProcess11 1.165185  
## 0 BiologicalMaterial01 1.147612  
## 37 ManufacturingProcess28 1.146329  
## 38 ManufacturingProcess29 1.063466  
## 40 ManufacturingProcess31 1.056690  
## 27 ManufacturingProcess18 1.038803  
## 13 ManufacturingProcess04 1.012270  
## 35 ManufacturingProcess26 0.991552  
## 39 ManufacturingProcess30 0.893542  
## 34 ManufacturingProcess25 0.890678  
## 14 ManufacturingProcess05 0.883578  
## 19 ManufacturingProcess10 0.883260  
## 8 BiologicalMaterial10 0.844341  
## 12 ManufacturingProcess02 0.836588  
## 29 ManufacturingProcess20 0.818080  
## 28 ManufacturingProcess19 0.804835  
## 44 ManufacturingProcess37 0.793354  
## 47 ManufacturingProcess42 0.764370  
## 48 ManufacturingProcess43 0.752079  
## 36 ManufacturingProcess27 0.732528  
## 43 ManufacturingProcess35 0.680018  
## 50 ManufacturingProcess45 0.644061  
## 30 ManufacturingProcess21 0.633637  
## 23 ManufacturingProcess14 0.630109  
## 24 ManufacturingProcess15 0.615345  
## 7 BiologicalMaterial09 0.606772  
## 4 BiologicalMaterial05 0.594744  
## 46 ManufacturingProcess39 0.584382  
## 49 ManufacturingProcess44 0.544347  
## 11 ManufacturingProcess01 0.521341  
## 16 ManufacturingProcess07 0.497345  
## 33 ManufacturingProcess24 0.471487  
## 32 ManufacturingProcess23 0.459346  
## 31 ManufacturingProcess22 0.364191  
## 17 ManufacturingProcess08 0.361650  
## 45 ManufacturingProcess38 0.350954  
## 25 ManufacturingProcess16 0.342756

### (f)

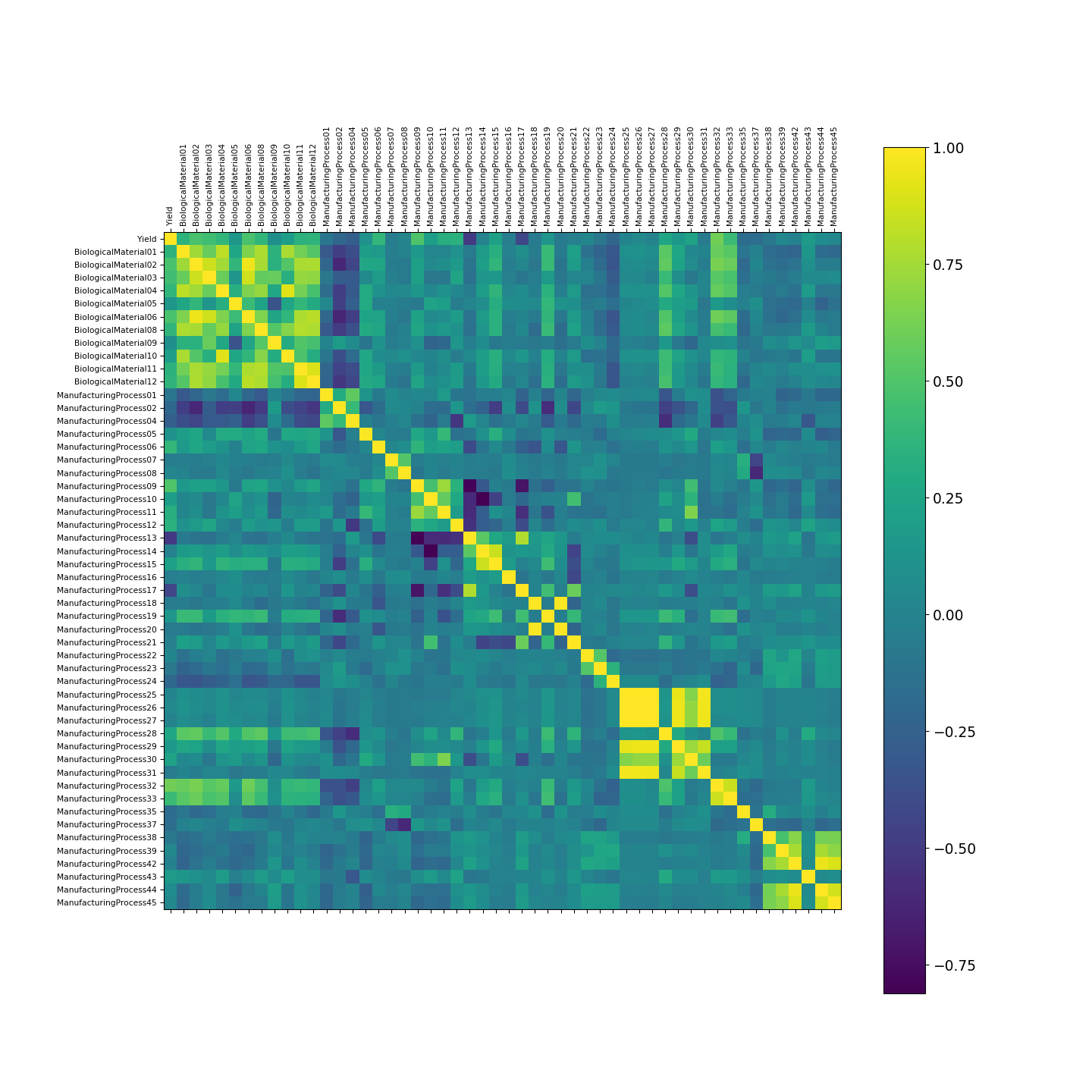
**Explore the relationships between each of the top predictors and the response. How could this information be helpful in improving yield in future runs of the manufacturing process?**

Lastly, we simply recreated the correlogram – just adding Yield so that we could see the relationship between the predictors and Yield. We find that the most important process ManufacturingProcess32 has a positive relationship with Yield, ManufacturingProcess13 has a negative relationship with Yield, and ManufactuingProcess09 has a slight positive relationship with Yield. We recommend that the manufacturing processes that have a positive relationship with Yield be increased and those that have a negative be decreased.

f = plt.figure(figsize=(15, 15))  
plt.matshow(pdf0var.corr(), fignum=f.number)  
plt.xticks(range(pdf0var.select\_dtypes(['number']).shape[1]), pdf0var.select\_dtypes(['number']).columns, fontsize=8, rotation=90)

plt.yticks(range(pdf0var.select\_dtypes(['number']).shape[1]), pdf0var.select\_dtypes(['number']).columns, fontsize=8)

cb = plt.colorbar()  
cb.ax.tick\_params(labelsize=14)  
plt.show()



# 7.2

**Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data:**

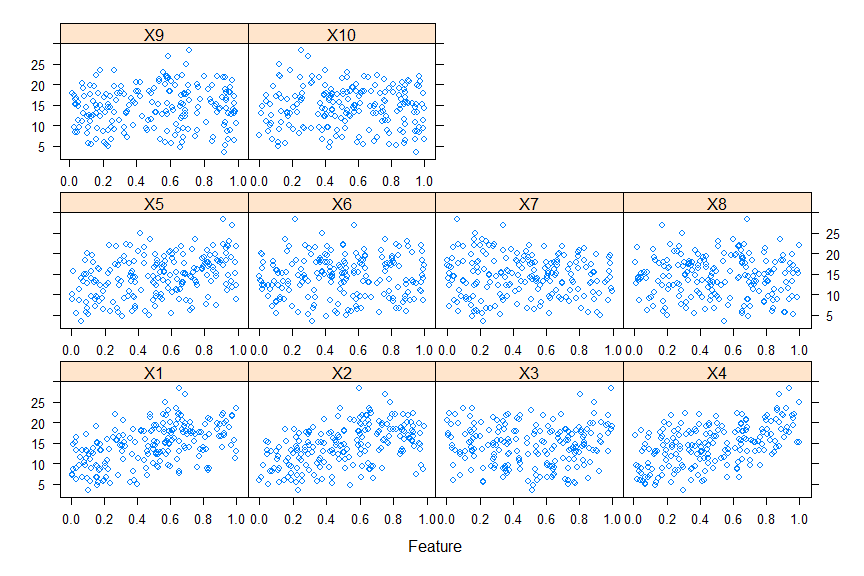
**y = 10 sin(πx1x2) + 20(x3 − 0.5)2 + 10x4 + 5x5 + N(0, σ2)**

**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)

## Warning: package 'mlbench' was built under R version 4.2.3

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)

**Tune several models on these data. For example:**

library(caret)  
knnModel <- train(x = trainingData$x,   
 y = trainingData$y,   
 method = "knn",  
 preProc = c("center", "scale"),  
 tuneLength = 10)  
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.

knnPred <- predict(knnModel, newdata = testData$x)  
## The function 'postResample' can be used to get the test set  
## perforamnce values  
postResample(pred = knnPred, obs = testData$y)

## RMSE Rsquared MAE   
## 3.2040595 0.6819919 2.5683461

**Which models appear to give the best performance? Does MARS select the informative predictors (those named X1–X5)?**

We used the training/test split that was created in R to train a KNN, SVM and Neural network models. The anaconda distribution does not have the pyearth library, which is what would be needed to attempt a MARS model. Comparing the three models that we did run, the SVM model was best, in R SVM was second best to MARS. KNN behaved almost identical to the R version and the neural net behaved slightly worse, likely because we did no tuning.

import numpy as np  
from sklearn.neighbors import KNeighborsRegressor  
knn\_model = KNeighborsRegressor(n\_neighbors=17)  
  
data\_train = r.trainingData.items()  
data\_train\_list = list(data\_train)  
data\_train\_np = np.array(data\_train\_list)

knn\_model.fit(data\_train\_np[0][1], data\_train\_np[1][1])

## KNeighborsRegressor(n\_neighbors=17)

from sklearn.metrics import mean\_squared\_error  
from math import sqrt  
  
data\_test = r.testData.items()  
data\_test\_list = list(data\_test)  
data\_test\_np = np.array(data\_test\_list)

knn\_preds = knn\_model.predict(data\_test\_np[0][1])  
mse = mean\_squared\_error(data\_test\_np[1][1], knn\_preds)  
rmse = sqrt(mse)  
rmse

## 3.2393682676309052

from sklearn.neural\_network import MLPRegressor  
  
nn\_model = MLPRegressor(random\_state=1, max\_iter=500).fit(data\_train\_np[0][1], data\_train\_np[1][1])

nn\_preds = nn\_model.predict(data\_test\_np[0][1])  
mse = mean\_squared\_error(data\_test\_np[1][1], nn\_preds)  
rmse = sqrt(mse)  
rmse

## 3.261108607338163

from sklearn import svm  
svm\_model = svm.SVR()  
svm\_model.fit(data\_train\_np[0][1], data\_train\_np[1][1])

## SVR()

svm\_preds = svm\_model.predict(data\_test\_np[0][1])  
mse = mean\_squared\_error(data\_test\_np[1][1], svm\_preds)  
rmse = sqrt(mse)  
rmse

## 2.922945911293997