# Assignment 3

*Nishanth Gandhidoss*

*10/12/2017*

# Question 1

## Section (a)

First let us take the data needed from the caret package. Using the code I have loaded the data and just to show/verify I have loaded the data properly, I am printin out a small subset of the data here.

### Subset of predictors and target variable

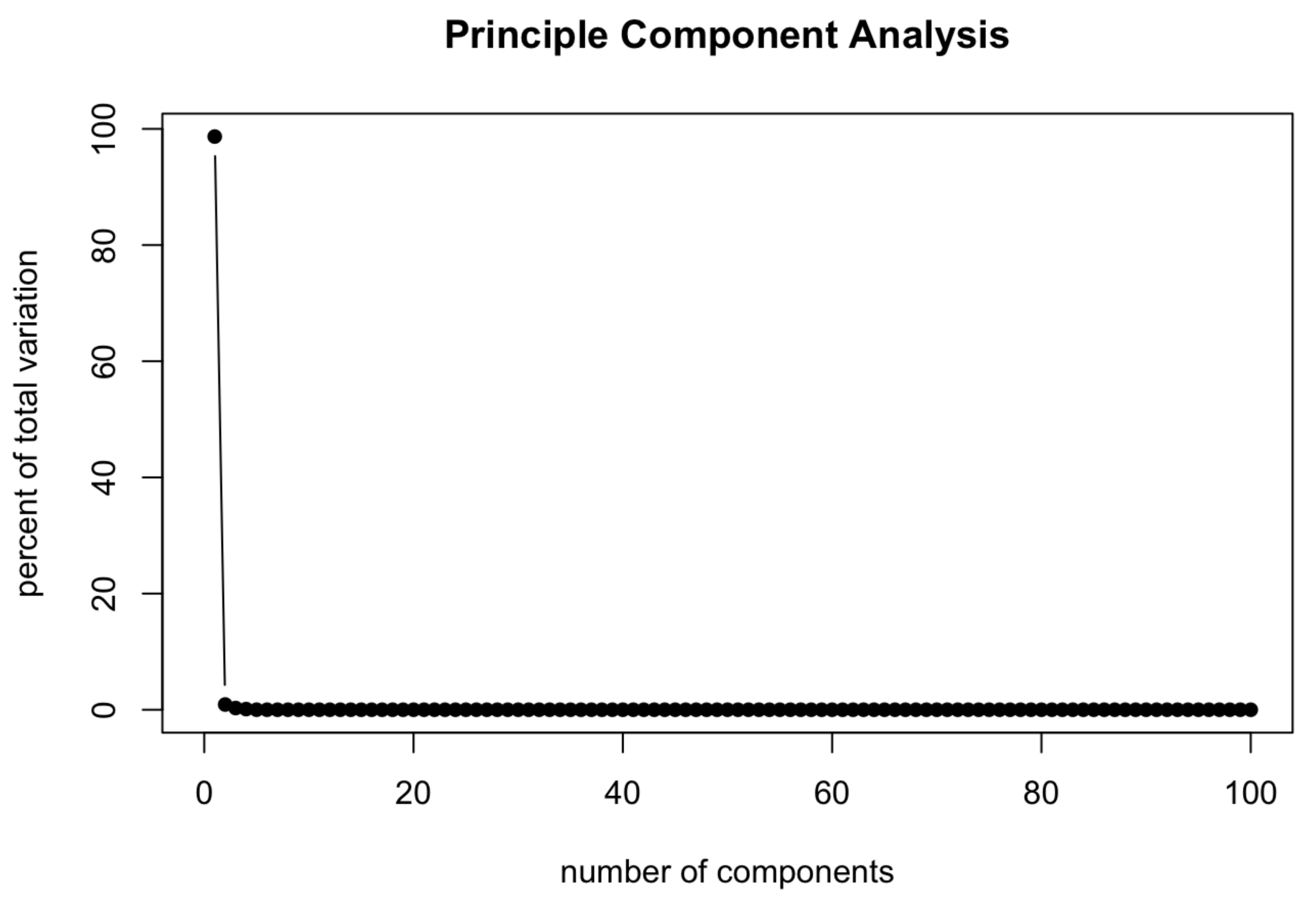
```
##        V1      V2      V3      V4      V5
## 1 2.61776 2.61814 2.61859 2.61912 2.61981
## 2 2.83454 2.83871 2.84283 2.84705 2.85138
## 3 2.58284 2.58458 2.58629 2.58808 2.58996
## 4 2.82286 2.82460 2.82630 2.82814 2.83001
## 5 2.78813 2.78989 2.79167 2.79350 2.79538
## 6 3.00993 3.01540 3.02086 3.02634 3.03190
```

```
##   mositure  fat protein
## 1     60.5 22.5    16.7
## 2     46.0 40.1    13.5
## 3     71.0  8.4    20.5
## 4     72.8  5.9    20.7
## 5     58.3 25.5    15.5
## 6     44.0 42.7    13.7
```

Thus we have above the data loaded properly. The dataframe absorp_df contains the 100 absorbance values for the 215 samples and the dataframe endpoints_df contains the percent of moisture, fat, and protein as the target varaibel for the predictors.

## Section (b)

Since the number of samples in the data is not very big compared to the number of predictor on comparision to most seen datasets in the world, let us use PCA to find the effective dimension of these data. This will help us to reduce the predictors according to the how much varaiblity we would like to explain with our model. I am using prcomp() to compute the Principal Componets



```
## [1] "Top 5 PCs"
```

```
## [1] 98.679162750  0.900926147  0.296292185  0.114005307  0.005754017
```

From the graph and the table, We can clearly see that the first component alone explains almost expains 98% of variablity in the data. Thus we clearly understand the true dimensionality of the data is much lower than the number of predictors in the data. So using PCA, we can say that the effective dimension for this data is only the first component. So we will reduce our predictors from 100 to 1. This says that using one predictors to make the prediction would be a better option.

## Section (c)

Before we generate the model we need to split the data into training and testing samples. Given the sample size, we will retain the 80% of the samples to the training set and 20% of the sample in the testing set. The train set will be used to tune the models by splitting that into 10 fold for cross validation in order to have better model performance. For spliting the train set we will use Leave group out cross validation with 5 folds.

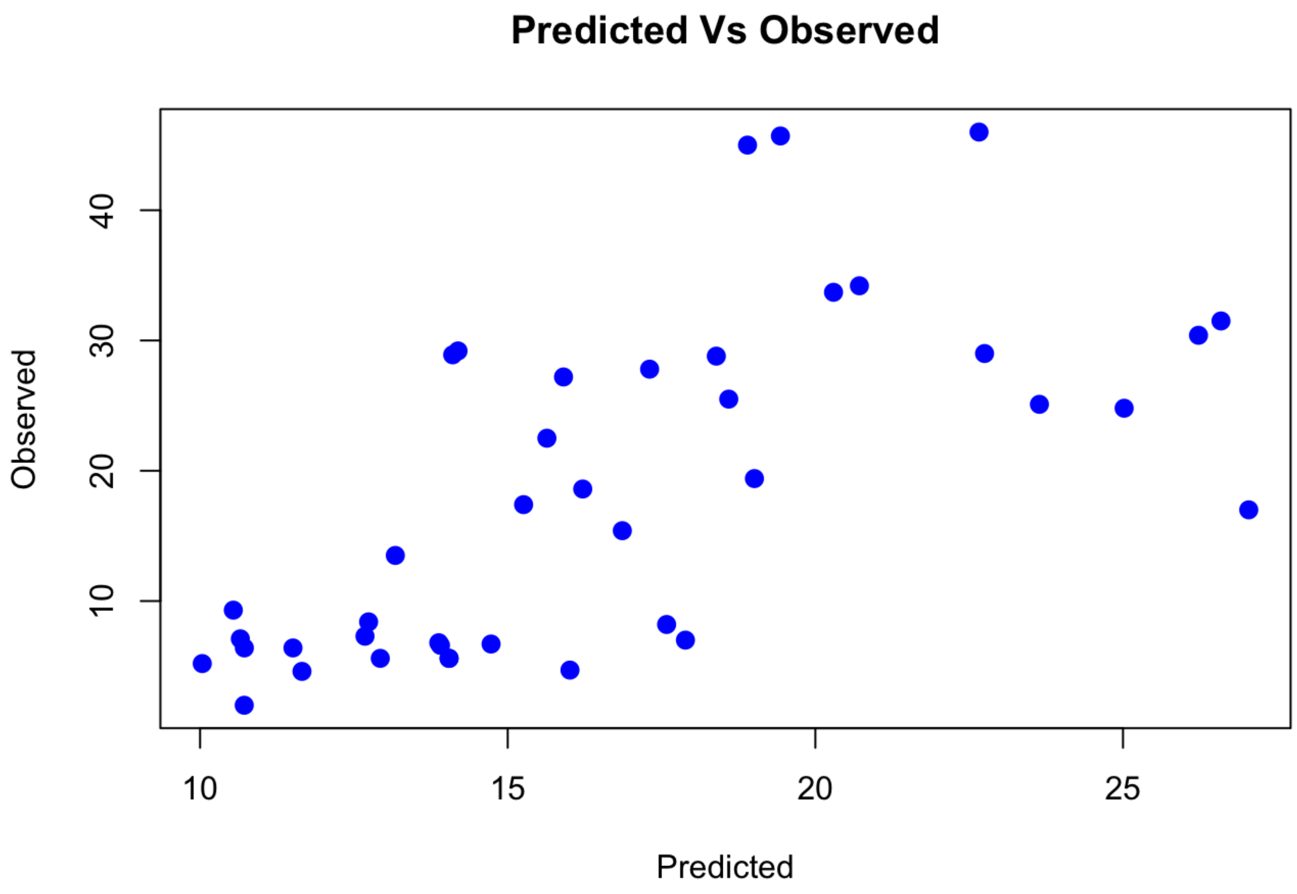
## Linear model

Thus after completing spoliting ouf the data, it is time to create models. First up is to create the linear models. Onething we need to think about before buliding the model is the correlation of our predictors. We will find the correlation and use a threshold value of .9 to identify and leeve the predictors which are tend to be highly skwed. In order to find the correlation I have used the findCorrelation() to do this.

```
## [1] "Not correlated predictors in the data is"
## [1] "V100"
```

Thus we will use only one predictors ie) V100 in our model that explains 98% of variablity. Lets use that predictors to bulid our linear model

```
## Linear Regression
##
## 174 samples
##   1 predictor
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 132, 132, 132, 132, 132, 132, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   11.54744  0.2421744  9.249985
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```



```
## [1] "Linear model has used only first predictor V100 in the model according to c
orrelation cut off 0.9"
```

```
## [1] "The RMSE value for linear model on the train set 11.5474"
```

```
## [1] "The R^2 value for linear model on the train set 0.2422"
## [1] "The RMSE value for linear model on the test set 10.2572"
```

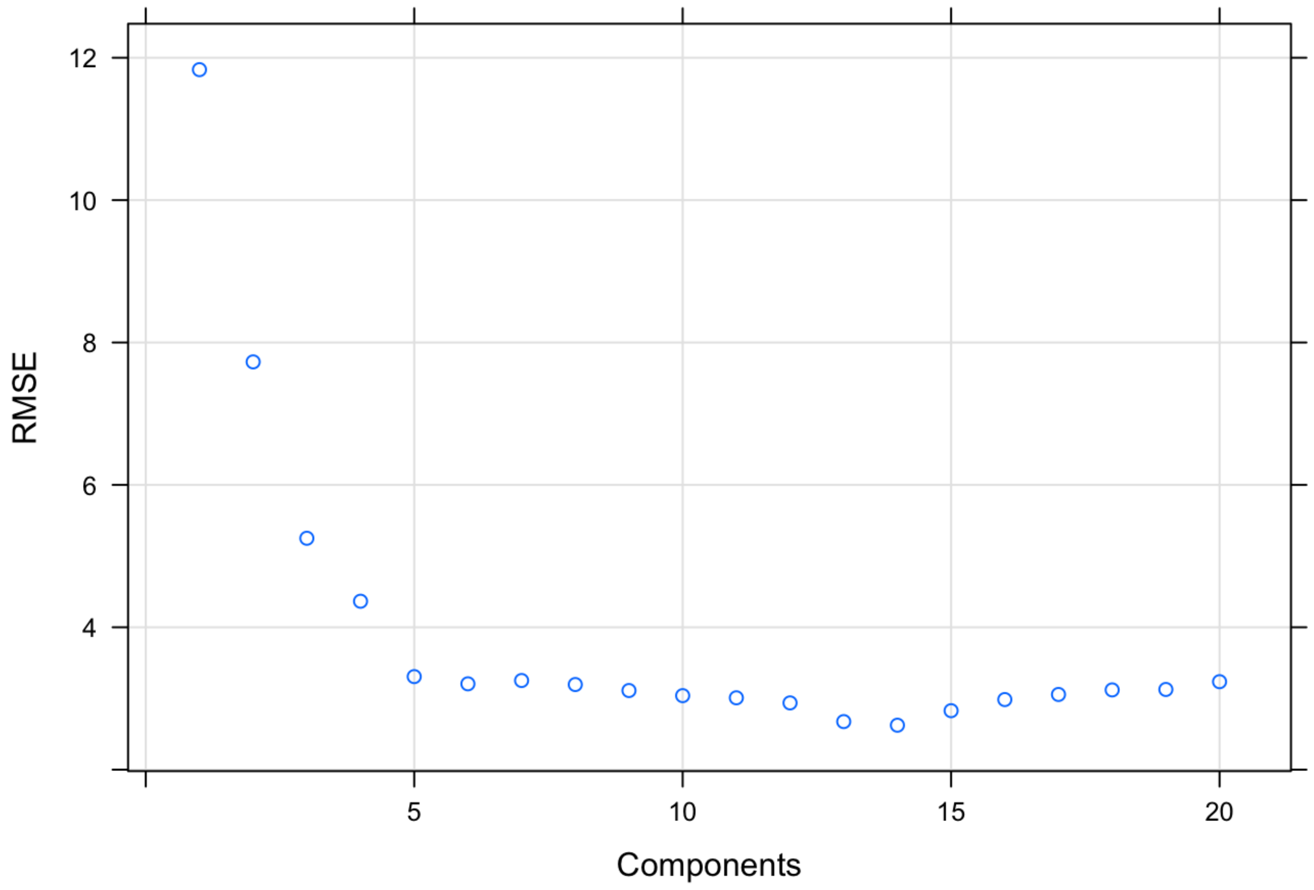
```
## [1] "The R^2 value for linear model on the test set 0.4361"
```

Thus the above plot shows the predicted vs observered plot of the linear model. And follwoed by we have the model's tuning parameter, RMSE and R square valeu of the train and test sets.

## Partial Least Square

Now let us use partial least square model on our dataset with all the predictors. Here we will use the train() in the caret package with method = pls that represents partial least square. And we will set the train control with our contraol grid. Preprocessing of the data is necessary for this model so we process the data using centering and scaling. After training the model we have the follwoing results.

```
## Partial Least Squares
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 132, 132, 132, 132, 132, 132, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##    1     11.832594  0.1994752  9.574986
##    2      7.728018  0.6602401  5.920319
##    3      5.250372  0.8370973  4.009136
##    4      4.365891  0.8922939  3.571557
##    5      3.307131  0.9425534  2.573182
##    6      3.205771  0.9453175  2.524375
##    7      3.252278  0.9440886  2.500562
##    8      3.195624  0.9459036  2.373648
##    9      3.111507  0.9475895  2.330050
##   10      3.039799  0.9504609  2.328043
##   11      3.009373  0.9519272  2.308444
##   12      2.937854  0.9539127  2.149050
##   13      2.675224  0.9618591  1.976754
##   14      2.624367  0.9628778  1.873714
##   15      2.828508  0.9564083  1.930094
##   16      2.984677  0.9507853  1.958088
##   17      3.056172  0.9491102  1.976901
##   18      3.120209  0.9466147  1.950940
##   19      3.126463  0.9457116  1.921792
##   20      3.235735  0.9419410  1.966429
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was ncomp = 14.
```



```
## [1] "The final value used for the model has 14 components"
```

```
## [1] "The RMSE value for pls model on the train set 2.6244"
```

```
## [1] "The R^2 value for pls model on the train set 0.9629"
```

```
## [1] "The RMSE value for pls model on the test set 2.2911"
```

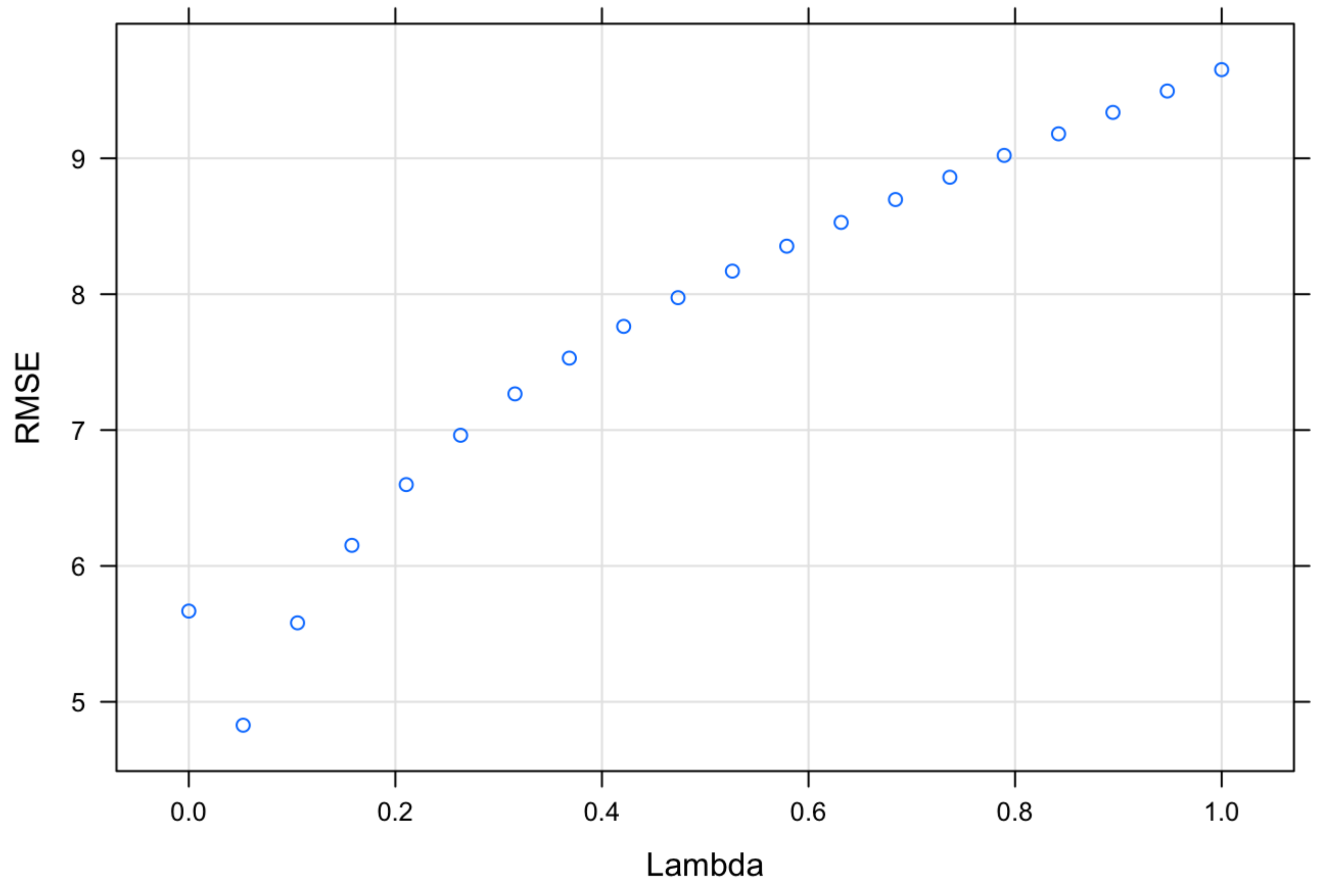
```
## [1] "The R^2 value for pls model on the test set 0.9669"
```

Thus the above plot shows that the 14 components is the best value to choose for the train set. And followed by we have the model's tuning parameter, RMSE and R square value of the train and test sets.

## Ridge Regression

Let us check how ridge regression is fitting the dataset. Here we will use the train() in the caret package with method = ridge that represents ridge regression. And we will set the train control with our control grid. Preprocessing of the data is necessary for this model so we process the data using centering and scaling. We will use 20 different values of lambda from 0 to 1. After training the model we have the follwoing results.

```
## Ridge Regression
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 132, 132, 132, 132, 132, 132, ...
## Resampling results across tuning parameters:
##
##   lambda      RMSE      Rsquared   MAE
##   0.00000000  5.667902  0.8483255  3.093479
##   0.05263158  4.827779  0.8821986  3.867906
##   0.10526316  5.581140  0.8503893  4.464085
##   0.15789474  6.151616  0.8181403  4.897354
##   0.21052632  6.598682  0.7861152  5.212610
##   0.26315789  6.961703  0.7551565  5.457527
##   0.31578947  7.266363  0.7257657  5.655017
##   0.36842105  7.529619  0.6981849  5.818806
##   0.42105263  7.762984  0.6724893  5.966196
##   0.47368421  7.974497  0.6486537  6.097616
##   0.52631579  8.169918  0.6265959  6.218264
##   0.57894737  8.353464  0.6062046  6.331822
##   0.63157895  8.528282  0.5873561  6.441090
##   0.68421053  8.696752  0.5699251  6.548291
##   0.73684211  8.860699  0.5537900  6.657056
##   0.78947368  9.021537  0.5388361  6.770057
##   0.84210526  9.180367  0.5249575  6.889864
##   0.89473684  9.338054  0.5120571  7.012920
##   0.94736842  9.495277  0.5000467  7.135749
##   1.00000000  9.652572  0.4888468  7.262752
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was lambda = 0.05263158.
```



```
## [1] "The final value used for the model has 0.0526315789473684 as lambda value"
```

```
## [1] "The RMSE value for ridge model on the train set 4.8278"
```

```
## [1] "The R^2 value for rigde model on the train set 0.8822"
```

```
## [1] "The RMSE value for ridge model on the test set 4.0576"
```

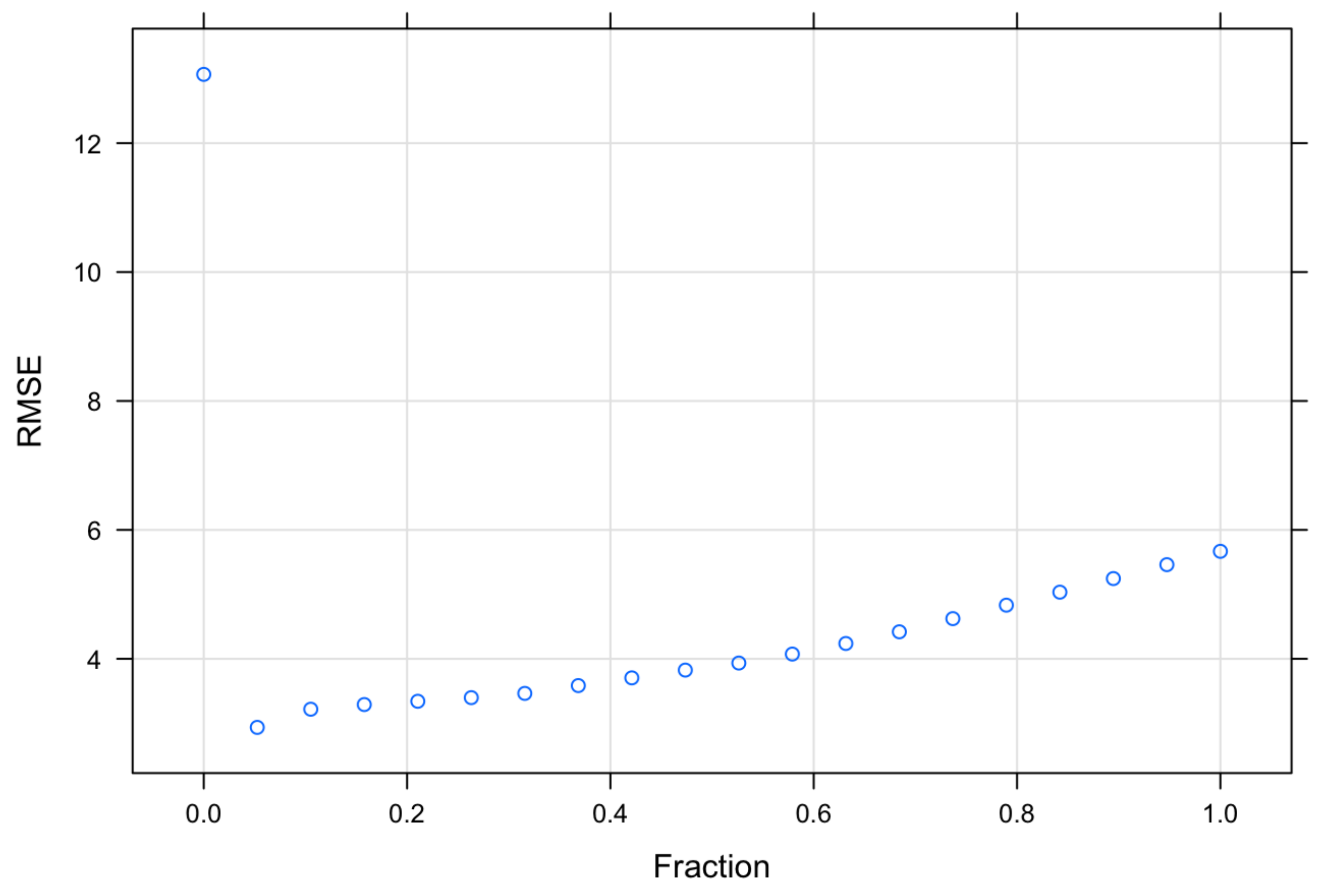
```
## [1] "The R^2 value for rigde model on the test set 0.9325"
```

Thus the above plot shows that the best lambda value for this dataset is 0.05263158 that is choosed from the train set using ridge regression. And followed by we have the model's tuning parameter, RMSE and R square value of the train and test sets.

## Lasso Regression

Let us try lasso regualizer for the dataset. Here we will be using the train() in the caret package with method = enet but that represents lasso regression by setting the lambda in the tune grid to be zero and varying the fraction parameter. Preprocessing of the data is necessary for this model so we process the data using centering and scaling. We will use 20 different values of fraction from 0 to 1. After training the model we have the follwoing results.

```
## Elasticnet
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 132, 132, 132, 132, 132, 132, ...
## Resampling results across tuning parameters:
##
##   fraction    RMSE       Rsquared   MAE
##   0.00000000  13.066953        NaN  10.926335
##   0.05263158   2.936291  0.9523524   1.822531
##   0.10526316   3.218014  0.9416768   1.915869
##   0.15789474   3.290396  0.9380186   1.948912
##   0.21052632   3.341285  0.9357111   1.969148
##   0.26315789   3.397666  0.9333807   1.996346
##   0.31578947   3.463579  0.9304378   2.043469
##   0.36842105   3.584759  0.9256353   2.111878
##   0.42105263   3.704386  0.9219933   2.176660
##   0.47368421   3.824965  0.9179387   2.239298
##   0.52631579   3.934127  0.9137985   2.294016
##   0.57894737   4.073486  0.9085881   2.368540
##   0.63157895   4.237677  0.9029851   2.454126
##   0.68421053   4.419089  0.8968896   2.539248
##   0.73684211   4.623332  0.8896066   2.630073
##   0.78947368   4.832190  0.8817547   2.720243
##   0.84210526   5.033883  0.8739133   2.811245
##   0.89473684   5.244962  0.8655609   2.904347
##   0.94736842   5.460621  0.8568680   2.999024
##   1.00000000   5.667902  0.8483255   3.093479
##
## Tuning parameter 'lambda' was held constant at a value of 0
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were fraction = 0.05263158 and
##  lambda = 0.
```



```
## [1] "The final value used for the model has 0.0526315789473684 as fraction value
"
```

```
## [1] "The RMSE value for lasso model on the train set 2.9363"
```

```
## [1] "The R^2 value for lasso model on the train set 0.9524"
```

```
## [1] "The RMSE value for lasso model on the test set 1.8923"
```

```
## [1] "The R^2 value for lasso model on the test set 0.9778"
```

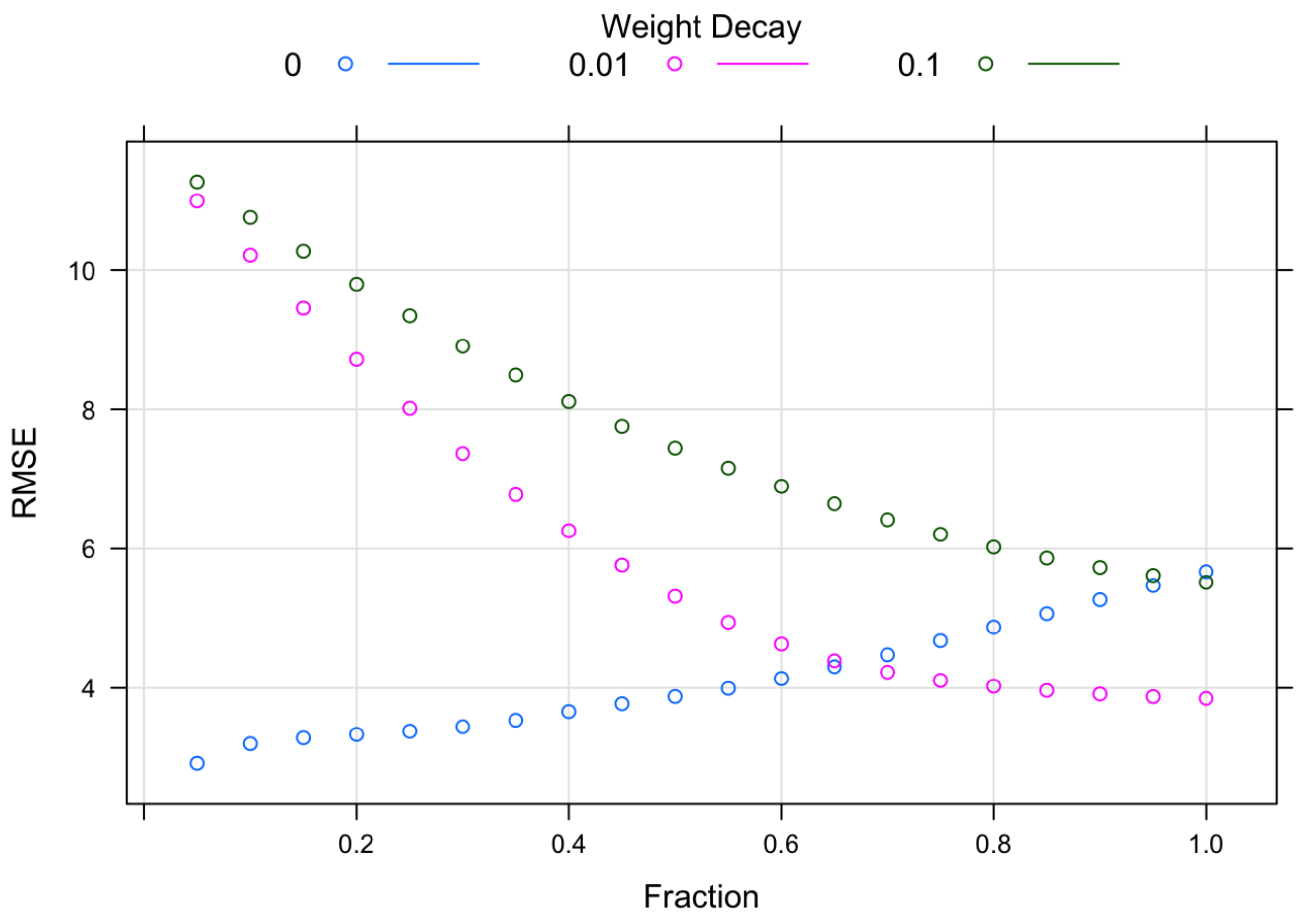
Thus the above plot shows that the best fraction value for this dataset is 0.05263158 that is choosed from the train set using lasso regression. And followed by we have the model's tuning parameter, RMSE and R square value of the train and test sets.

## Elastic Net

Let us try Elastic net regualizer for the dataset. Here we will be using the train() in the caret package with method = enet but by setting the lambda and fraction in the tune grid to be varying values so that it gives us the otimal parameter for the elastic net model. Preprocessing of the data is necessary for this model so we process the data using centering and scaling. We will use 20 different values of fraction from 0.05 to 1 and three different lambda value as 0, 0.01, 0.1. After training the model we have the following results.

```
## Elasticnet
##
```

```
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 132, 132, 132, 132, 132, 132, ...
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE       Rsquared   MAE
##   0.00    0.05       2.919209  0.9529798  1.819027
##   0.00    0.10       3.199890  0.9424071  1.908647
##   0.00    0.15       3.283666  0.9383745  1.947768
##   0.00    0.20       3.332206  0.9362228  1.964075
##   0.00    0.25       3.377766  0.9340971  1.986720
##   0.00    0.30       3.442306  0.9314791  2.027441
##   0.00    0.35       3.534832  0.9274420  2.085762
##   0.00    0.40       3.659306  0.9233706  2.152901
##   0.00    0.45       3.772169  0.9198373  2.212330
##   0.00    0.50       3.876307  0.9160095  2.265365
##   0.00    0.55       3.993671  0.9115049  2.324006
##   0.00    0.60       4.133848  0.9064804  2.401442
##   0.00    0.65       4.302987  0.9008088  2.484827
##   0.00    0.70       4.474453  0.8949685  2.564530
##   0.00    0.75       4.678346  0.8875847  2.654101
##   0.00    0.80       4.873163  0.8801577  2.737881
##   0.00    0.85       5.065406  0.8726685  2.825384
##   0.00    0.90       5.266976  0.8646759  2.914056
##   0.00    0.95       5.470857  0.8564480  3.003655
##   0.00    1.00       5.667902  0.8483255  3.093479
##   0.01    0.05      10.992680  0.3380565  8.934464
##   0.01    0.10      10.210937  0.4724069  8.311047
##   0.01    0.15       9.452667  0.5884312  7.711994
##   0.01    0.20       8.717552  0.6796589  7.122010
##   0.01    0.25       8.014693  0.7462907  6.546910
##   0.01    0.30       7.362510  0.7919515  6.004774
##   0.01    0.35       6.775711  0.8222695  5.506341
##   0.01    0.40       6.256411  0.8442106  5.068405
##   0.01    0.45       5.764166  0.8641760  4.663961
##   0.01    0.50       5.315605  0.8802269  4.293464
##   0.01    0.55       4.941679  0.8914603  3.970314
##   0.01    0.60       4.629488  0.8997011  3.723757
##   0.01    0.65       4.389054  0.9053518  3.556227
##   0.01    0.70       4.224757  0.9089099  3.450780
##   0.01    0.75       4.106871  0.9115611  3.372763
##   0.01    0.80       4.024840  0.9135825  3.316666
##   0.01    0.85       3.963882  0.9152812  3.273306
##   0.01    0.90       3.913473  0.9170277  3.234291
##   0.01    0.95       3.874899  0.9184620  3.203342
##   0.01    1.00       3.848966  0.9194849  3.181553
##   0.10    0.05      11.263993  0.2825399  9.125926
##   0.10    0.10      10.756669  0.3643711  8.723472
##   0.10    0.15      10.267399  0.4432773  8.336636
##   0.10    0.20       9.796545  0.5153644  7.965257
##   0.10    0.25       9.343151  0.5788627  7.603797
##   0.10    0.30       8.907243  0.6331298  7.250699
##   0.10    0.35       8.494620  0.6779273  6.913429
##   0.10    0.40       8.110224  0.7138133  6.595848
##    0.10    0.45       7.757114  0.7417030  6.301273
##    0.10    0.50       7.440309  0.7632680  6.034836
##    0.10    0.55       7.154734  0.7810673  5.799062
##    0.10    0.60       6.894079  0.7963494  5.587637
##    0.10    0.65       6.644329  0.8103053  5.388292
##    0.10    0.70       6.413358  0.8219403  5.200412
##    0.10    0.75       6.205321  0.8312002  5.025910
##    0.10    0.80       6.022814  0.8383672  4.870067
##    0.10    0.85       5.864760  0.8438443  4.733198
##    0.10    0.90       5.728112  0.8480427  4.610443
##    0.10    0.95       5.614024  0.8511568  4.505521
##    0.10    1.00       5.515246  0.8535871  4.411174
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were fraction = 0.05 and lambda = 0.
```



```
## [1] "The final value used for the model has 0 as lambda value and 0.05 as fracti
on value"
```

```
## [1] "The RMSE value for elastic net model on the train set 2.9192"
```

```
## [1] "The R^2 value for elastic net model on the train set 0.953"
```

```
## [1] "The RMSE value for elastic net model on the test set 1.9004"
## [1] "The R^2 value for elastic net model on the test set 0.9776"
```

Thus the above plot shows that the best enet model has the parameter of fraction value as 0.05 and lambda as 0 that is choosed from the train set using Elastic net. And followed by we have the model's tuning parameter, RMSE and R square value of the train and test sets.

## Overall Results

After doing all five models the below table shows the Best parameter, RMSE and R square value of the Training and test set.

| **Model** | **Parameter** | **Training RMSE** | **Training R Squared** | **Testing RMSE** | **Testing R Squared** |
|---|---|---|---|---|---|
| Linear | Only V100 | 11.5474 | 0.2422 | 10.2572 | 0.4361 |
| PLS | 14 components | 2.6244 | 0.9629 | 2.2911 | 0.9669 |
| Ridge | Lambda as 0.05263158 | 4.7302 | 0.8804 | 4.0576 | 0.9325 |
| Lasso | Fraction as 0.05263158 | 2.9363 | 0.9524 | 1.8923 | 0.9778 |
| Elastic Net | lambda as 0 & Fraction as 0.05 | 2.9192 | 0.953 | 1.9004 | 0.9776 |

Thus from the table, the optimal tuning parameter value for each model can be found in the Parameters column. The best tunning parameter found from the training set is used on the testing set to make prediction on it for each models. And that is how we find the RMSE and R square value for the test set.

## Section (d)

According to the above results in the table we can say that lasso model has the best predictive ability because it has highest R square value above all other results. On looking at the R square and RMSE value, linear model is significantly worst than other models.

## Section (e)

I would use Lasso for predicting the fat content of a sample as the R squared value on the test set for it 0.9778 which is very large.

# Question 2

## Section (a)

We will first load the dataset here in order to answer the further question. To ensure I have the data I checked the data dimension and printed out a small subset of the data.

```
##   X1 X2 X3 X4 X5
## 1  0  0  0  0  0
## 2  0  0  0  0  0
## 3  0  0  0  0  0
## 4  0  0  0  0  0
## 5  0  0  0  0  0
## 6  0  0  0  0  0
```

```
## [1] 12.520  1.120 19.405  1.730  1.680
```

Thus we see that both the predictors and target values are loaded properly. The fingerprints contains the 1107 binary molecular predictors for the 165 compounds, while permeability contains permeability response is the responding variable.

## Section (b)

It is been told that the fingerprint predictors are having sparse data. In order to handle this we are going to apply near zero variance to handle this using caret package.

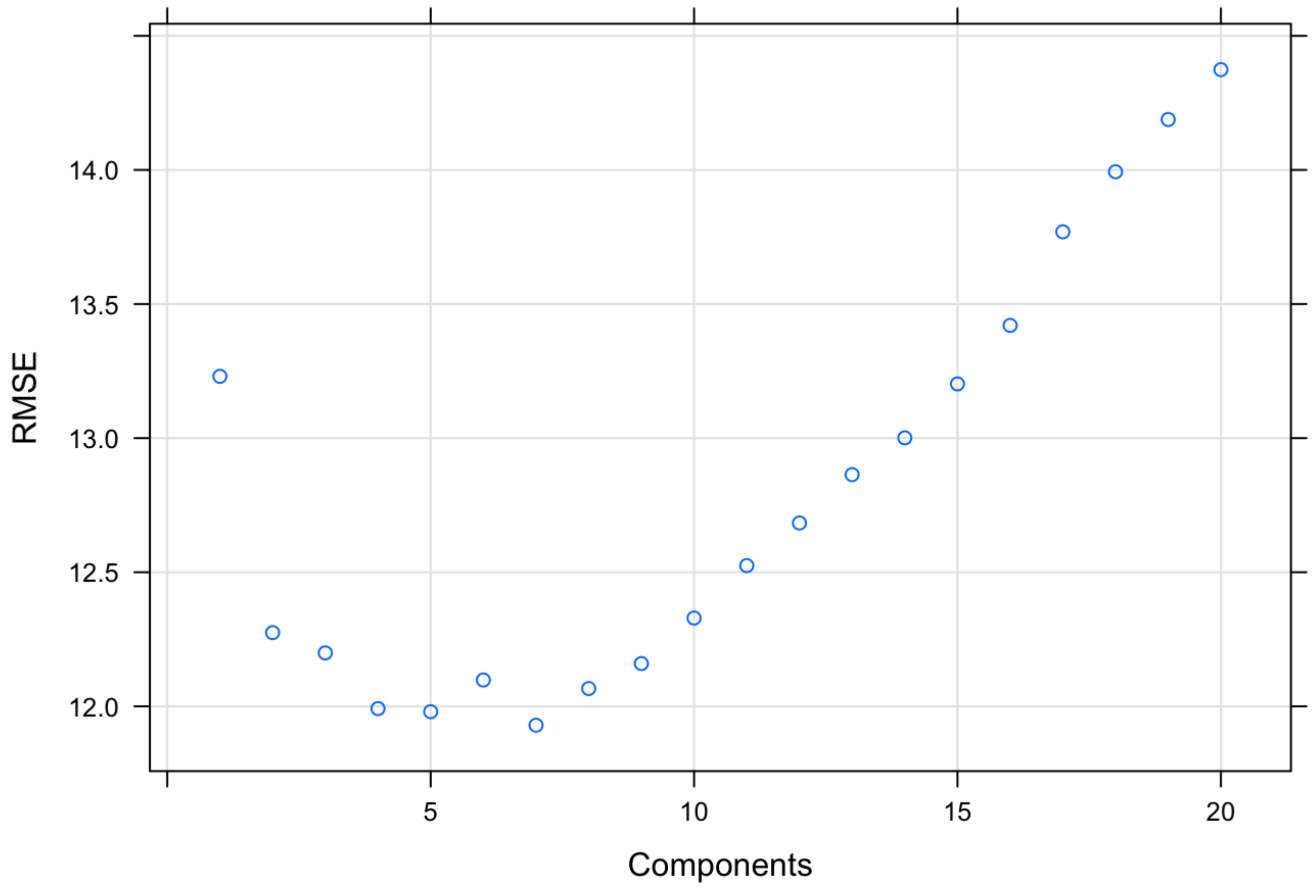
```
## [1] "Number of predictors left out for modeling is 388"
```

Thus after removing the near zero variance from the model we have 388 predictors left our in the data. We will use this predictors to bulid the models.

## Section (c)

To split the data into a training and a test set, I have 80% of the data put into the train set and left out data samples in the train set. Then using the splitted train set, I willuse Leave group out cross validation because the size of the data points is considerably greater than the size of the predictors. To bulid the model, I have used train() in the caret package and preprocessed the data using center and scaling to tune a partial least squares model.

```
## Partial Least Squares
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##    1     13.23038  0.3047124   9.892392
##    2     12.27477  0.4251030   8.825577
##    3     12.19916  0.4320105   9.325554
##    4     11.99150  0.4564663   8.991065
##    5     11.97992  0.4697041   8.869150
##    6     12.09807  0.4614077   8.909500
##    7     11.92951  0.4759474   8.850921
##    8     12.06621  0.4743570   9.125090
##    9     12.15962  0.4751014   9.085340
##   10     12.32913  0.4682292   9.180052
##   11     12.52472  0.4579571   9.262803
##   12     12.68360  0.4514604   9.463243
##   13     12.86389  0.4460969   9.561076
##   14     13.00112  0.4401731   9.690260
##   15     13.20214  0.4317887   9.805475
##   16     13.42036  0.4249043   9.984063
##   17     13.76940  0.4113484  10.245288
##   18     13.99321  0.4027498  10.434493
##   19     14.18815  0.3965162  10.499240
##   20     14.37402  0.3909217  10.573241
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was ncomp = 7.
```



```
## [1] "The final value used for the model has 7 components"
```

```
## [1] "The RMSE value for pls model on the train set 11.9295"
```

```
## [1] "The R^2 value for pls model on the train set 0.4759"
```

```
## [1] "The RMSE value for pls model on the test set 10.8979"
```

```
## [1] "The R^2 value for pls model on the test set 0.4457"
```

Above plot and the results shows that using 7 components would be optimal for this dataset with R squared value of 0.4759. In addition, I also have other information such as train and test set RMSE and R square value.

## Section (d)

The response for this test set using Partial Least Square value is as follows.

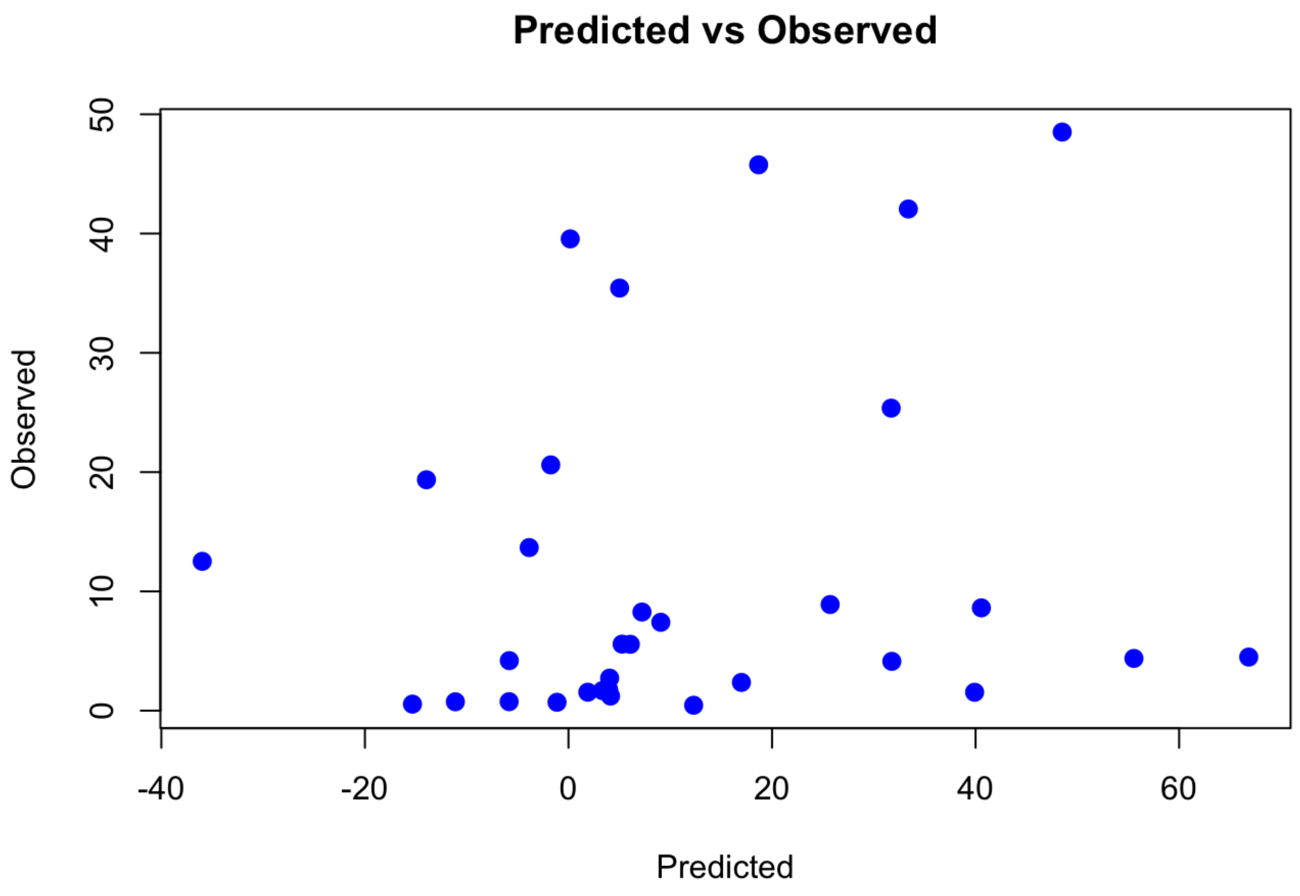
```
##  [1]  9.2339854 37.4838533  4.6546697 10.9737573  6.0174087  9.9936195
##  [7]  8.8063842  5.2640503  6.4010953 -0.1563235  4.9939793  1.3858175
## [13]  2.1129639 25.0880950  6.1492436 21.4133810 21.2280231 10.1224044
## [19] 17.5221565 28.7874372 23.1042341 10.6478795 38.1501484  3.6477972
## [25] -2.7284845  6.9206478 17.4437284 28.5749314  4.0442280  7.1495931
## [31]  1.3686449 27.9763206
```

The r squared value for the test set is 0.4457.

## Section (e)

## Linear model

Let us first bulid the linear model for this dataset. Usually for the continuous predictors we will be using correlation to process the data. But here we have categorical predictors which we have already been removed with near zero variance processing. I have used train() in the caret package using method as lm. With this we have the following results.



```
## [1] "The RMSE value for linear model on the train set 47.2918"
```

```
## [1] "The R^2 value for linear model on the train set 0.1072"
```

```
## [1] "The RMSE value for linear model on the test set 24.0788"
```

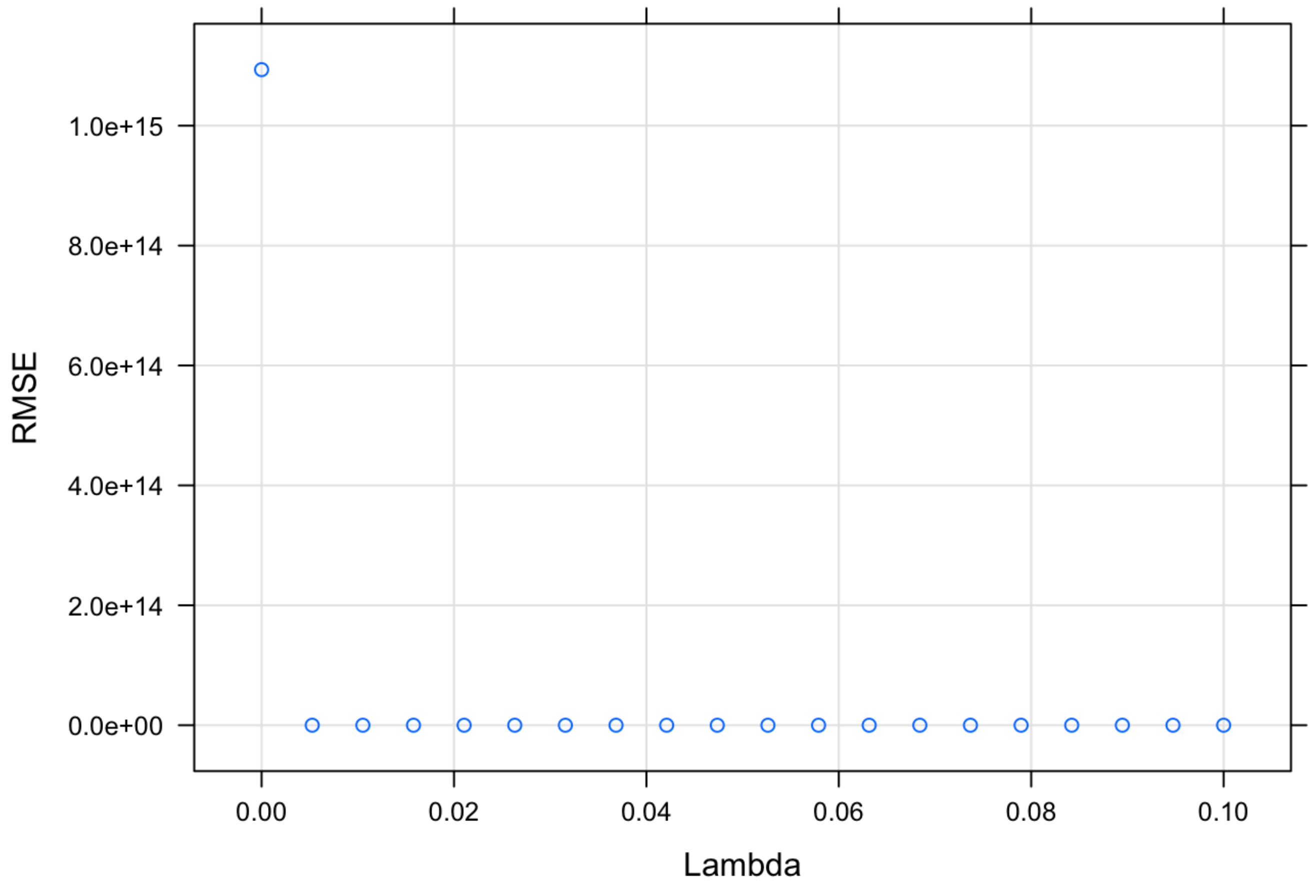
```
## [1] "The R^2 value for linear model on the test set 0.0325"
```

Above results and the plot shows information such as train and test set RMSE and R square value of the linear model. This model looks worst as it has vert low R square value of 0.0325.

## Ridge Regression

Let us check how ridge regression is fitting the dataset. Here we will use the train() in the caret package with method = ridge that represents ridge regression. And we will set the train control with our control grid. Preprocessing of the data is necessary for this model so we process the data using centering and scaling. We will use 20 different values of lambda from 0 to 0.3 beacuse of the step raise if we set 0 to 1. After training the model we have the follwoing results.

```
## Ridge Regression
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   lambda       RMSE          Rsquared   MAE
##   0.000000000  1.093458e+15  0.3545942  3.865958e+14
##   0.005263158  6.383415e+02  0.2760523  4.042867e+02
##   0.010526316  3.503994e+01  0.3244908  2.357243e+01
##   0.015789474  7.812640e+01  0.3519869  4.433781e+01
##   0.021052632  1.475743e+01  0.3845154  1.089529e+01
##   0.026315789  1.433582e+01  0.3984241  1.058190e+01
##   0.031578947  1.444119e+01  0.3974814  1.066573e+01
##   0.036842105  1.389191e+01  0.4150783  1.024716e+01
##   0.042105263  1.370908e+01  0.4226190  1.010485e+01
##   0.047368421  1.364509e+01  0.4267986  1.007233e+01
##   0.052631579  1.343525e+01  0.4345104  9.910938e+00
##   0.057894737  1.335140e+01  0.4385038  9.850913e+00
##   0.063157895  1.326190e+01  0.4425646  9.790300e+00
##   0.068421053  1.319494e+01  0.4462418  9.741837e+00
##   0.073684211  1.313260e+01  0.4495026  9.699365e+00
##   0.078947368  1.308102e+01  0.4524087  9.664724e+00
##   0.084210526  1.305122e+01  0.4543228  9.645442e+00
##   0.089473684  1.300196e+01  0.4572349  9.609014e+00
##   0.094736842  1.295611e+01  0.4598426  9.573880e+00
##   0.100000000  1.292751e+01  0.4619990  9.552861e+00
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was lambda = 0.1.
```



```
## [1] "The final value used for the model has 0.1 as lambda value"
```

```
## [1] "The RMSE value for ridge model on the train set 12.9275"
```

```
## [1] "The R^2 value for rigde model on the train set 0.462"
```

```
## [1] "The RMSE value for ridge model on the test set 12.1947"
```

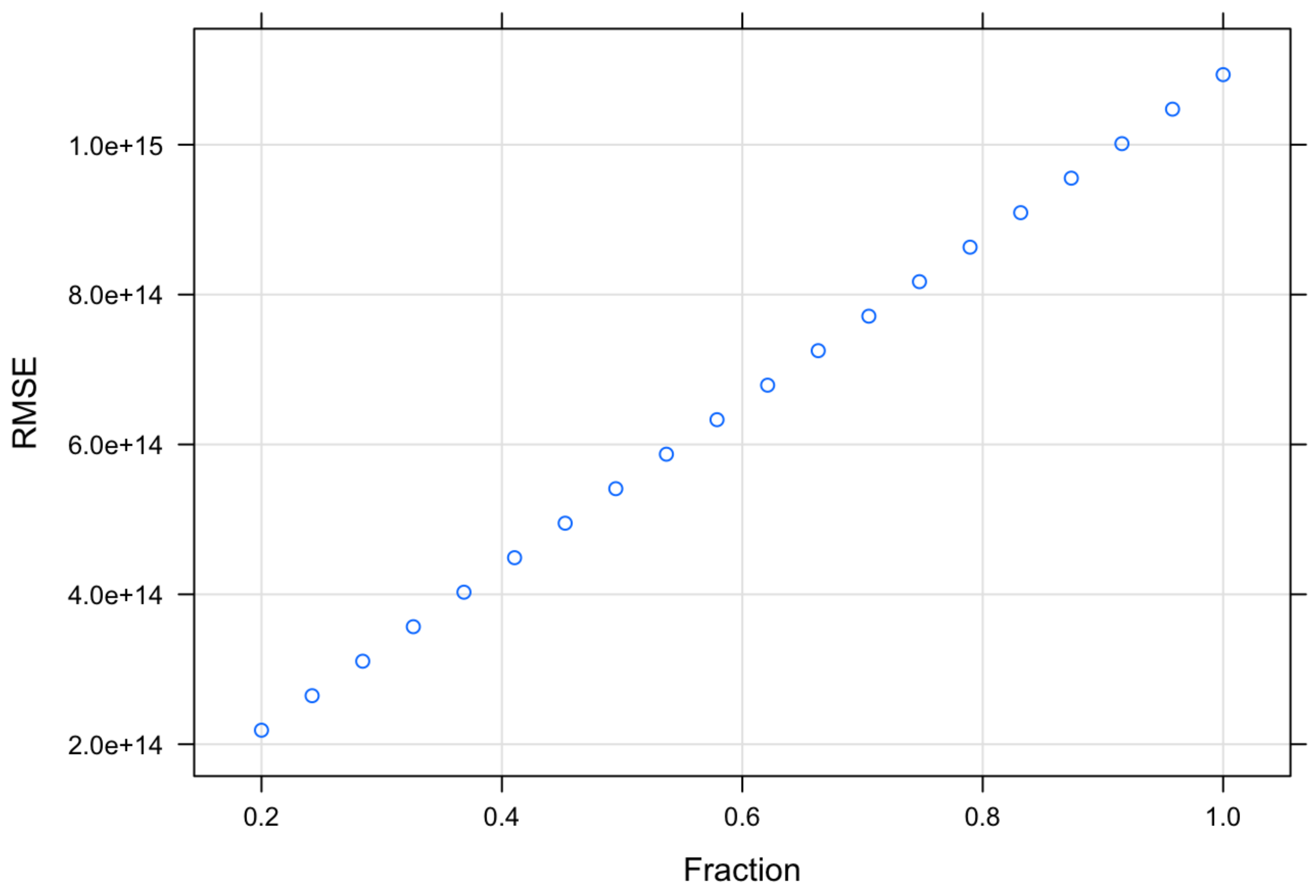
```
## [1] "The R^2 value for rigde model on the test set 0.385"
```

Thus the above plot shows that the best lambda value for this dataset is 0.1578947 that is choosed from the train set using ridge regression. And followed by we have the model's tuning parameter, RMSE and R square value of the train and test sets.

## Lasso Regression

Let us try lasso regualizer for the dataset. Here we will be using the train() in the caret package with method = enet but that represents lasso regression by setting the lambda in the tune grid to be zero and varying the fraction parameter. Preprocessing of the data is necessary for this model so we process the data using centering and scaling. We will use 20 different values of fraction from 0 to 1. After training the model we have the follwoing results.

```
## The lasso
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   fraction   RMSE          Rsquared   MAE
##   0.2000000  2.186916e+14  0.3857915  7.731915e+13
##   0.2421053  2.647319e+14  0.3845956  9.359687e+13
##   0.2842105  3.107723e+14  0.3840036  1.098746e+14
##   0.3263158  3.568126e+14  0.3849795  1.261523e+14
##   0.3684211  4.028529e+14  0.3838320  1.424300e+14
##   0.4105263  4.488933e+14  0.3842763  1.587077e+14
##   0.4526316  4.949336e+14  0.3839033  1.749854e+14
##   0.4947368  5.409739e+14  0.3828834  1.912632e+14
##   0.5368421  5.870143e+14  0.3807549  2.075409e+14
##   0.5789474  6.330546e+14  0.3781480  2.238186e+14
##   0.6210526  6.790949e+14  0.3761312  2.400963e+14
##   0.6631579  7.251353e+14  0.3746386  2.563740e+14
##   0.7052632  7.711756e+14  0.3736825  2.726517e+14
##   0.7473684  8.172159e+14  0.3714621  2.889295e+14
##   0.7894737  8.632563e+14  0.3690344  3.052072e+14
##   0.8315789  9.092966e+14  0.3665716  3.214849e+14
##   0.8736842  9.553369e+14  0.3639637  3.377626e+14
##   0.9157895  1.001377e+15  0.3608952  3.540403e+14
##   0.9578947  1.047418e+15  0.3578008  3.703180e+14
##   1.0000000  1.093458e+15  0.3545942  3.865958e+14
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was fraction = 0.2.
```



```
## [1] "The final value used for the model has 0.2 as fraction value"
```

```
## [1] "The RMSE value for lasso model on the train set 218691584763354"
```

```
## [1] "The R^2 value for lasso model on the train set 0.3858"
```

```
## [1] "The RMSE value for lasso model on the test set 11.4899"
```

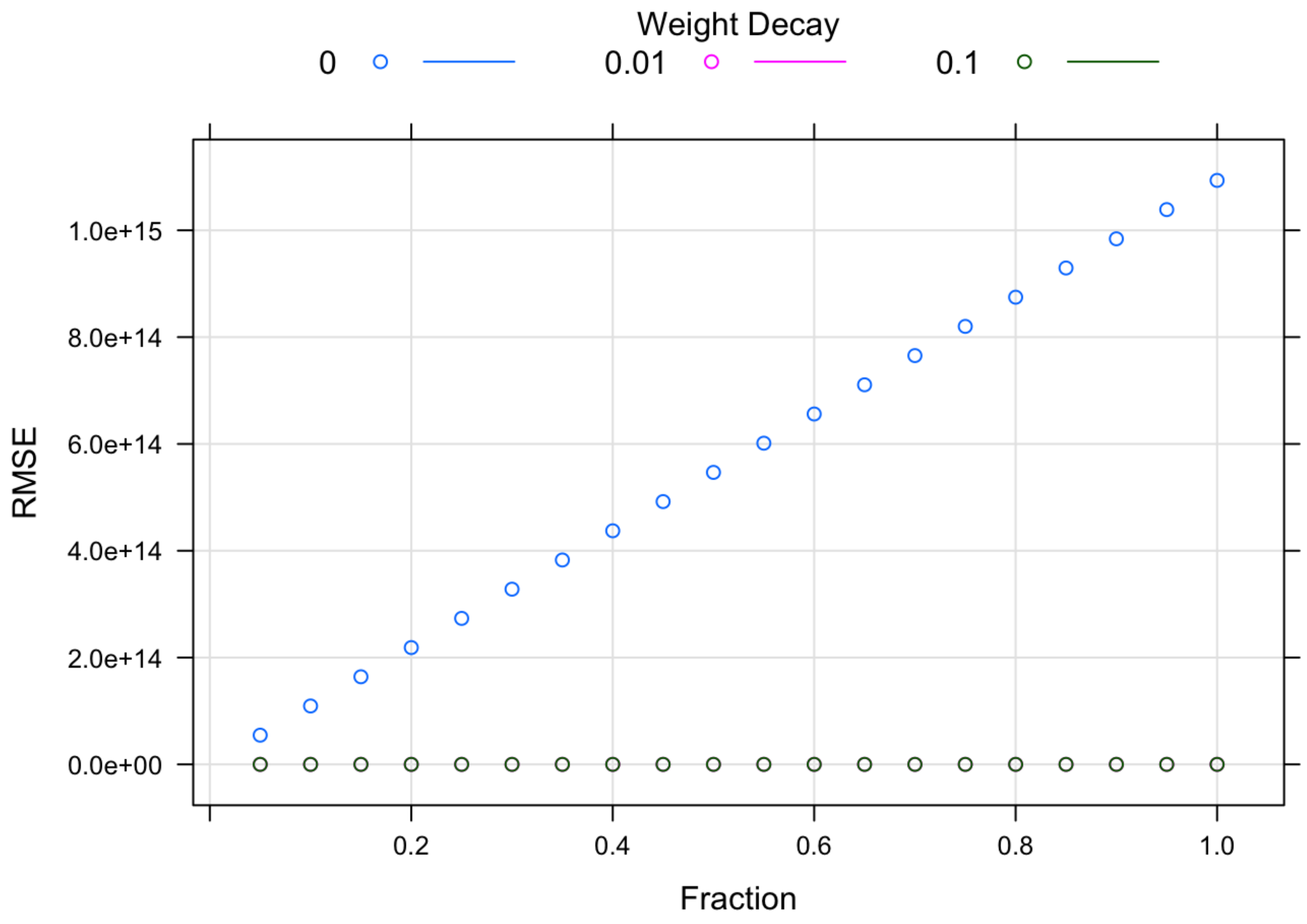
```
## [1] "The R^2 value for lasso model on the test set 0.3907"
```

Thus the above plot shows that the best fraction value for this dataset is that is choosed from the train set using lasso regression. And followed by we have the model’s tuning parameter, RMSE and R square value of the train and test sets.

## Elastic Net

Let us try Elastic net regualizer for the dataset. Here we will be using the train() in the caret package with method = enet but by setting the lambda and fraction in the tune grid to be varying values so that it gives us the otimal parameter for the elastic net model. Preprocessing of the data is necessary for this model so we process the data using centering and scaling. We will use 20 different values of fraction from 0.05 to 1 and three different lambda value as 0, 0.01, 0.1. After training the model we have the following results.

```
## Elasticnet
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE          Rsquared   MAE
##   0.00    0.05      5.467290e+13  0.3718045  1.932979e+13
##   0.00    0.10      1.093458e+14  0.3865474  3.865958e+13
##   0.00    0.15      1.640187e+14  0.3875214  5.798936e+13
##   0.00    0.20      2.186916e+14  0.3857915  7.731915e+13
##   0.00    0.25      2.733645e+14  0.3841484  9.664894e+13
##   0.00    0.30      3.280374e+14  0.3843900  1.159787e+14
##   0.00    0.35      3.827103e+14  0.3845748  1.353085e+14
##   0.00    0.40      4.373832e+14  0.3840195  1.546383e+14
##   0.00    0.45      4.920561e+14  0.3839480  1.739681e+14
##   0.00    0.50      5.467290e+14  0.3826403  1.932979e+14
##   0.00    0.55      6.014019e+14  0.3798850  2.126277e+14
##   0.00    0.60      6.560748e+14  0.3770037  2.319575e+14
##   0.00    0.65      7.107477e+14  0.3749687  2.512872e+14
##   0.00    0.70      7.654205e+14  0.3738934  2.706170e+14
##   0.00    0.75      8.200934e+14  0.3713122  2.899468e+14
##   0.00    0.80      8.747663e+14  0.3683568  3.092766e+14
##   0.00    0.85      9.294392e+14  0.3655224  3.286064e+14
##   0.00    0.90      9.841121e+14  0.3620678  3.479362e+14
##   0.00    0.95      1.038785e+15  0.3583976  3.672660e+14
##   0.00    1.00      1.093458e+15  0.3545942  3.865958e+14
##   0.01    0.05      5.510495e+01  0.3770137  3.609612e+01
##   0.01    0.10      9.775935e+01  0.3772914  6.211572e+01
##   0.01    0.15      1.404117e+02  0.3883983  8.840459e+01
##   0.01    0.20      1.830032e+02  0.3975189  1.145466e+02
##   0.01    0.25      2.236151e+02  0.3985496  1.398728e+02
##   0.01    0.30      2.633739e+02  0.3903525  1.653268e+02
##   0.01    0.35      3.032354e+02  0.3775500  1.908078e+02
##   0.01    0.40      3.431838e+02  0.3625201  2.163303e+02
##   0.01    0.45      3.831015e+02  0.3486766  2.418398e+02
##   0.01    0.50      4.226600e+02  0.3365182  2.671957e+02
##   0.01    0.55      4.618652e+02  0.3254289  2.923896e+02
##   0.01    0.60      5.010596e+02  0.3155651  3.175896e+02
##   0.01    0.65      5.402284e+02  0.3068768  3.427640e+02
##   0.01    0.70      5.793638e+02  0.2992554  3.679098e+02
##   0.01    0.75      6.184753e+02  0.2927527  3.930365e+02
##   0.01    0.80      6.575987e+02  0.2874164  4.181513e+02
##   0.01    0.85      6.967014e+02  0.2824084  4.432919e+02
##   0.01    0.90      7.357864e+02  0.2779471  4.684360e+02
##   0.01    0.95      7.748558e+02  0.2738843  4.935689e+02
##   0.01    1.00      8.134076e+02  0.2703748  5.182789e+02
##   0.10    0.05      1.261692e+01  0.4122044  9.469311e+00
##   0.10    0.10      1.208255e+01  0.4287547  8.567391e+00
##   0.10    0.15      1.211357e+01  0.4325814  8.583505e+00
##   0.10    0.20      1.200291e+01  0.4461844  8.627831e+00
##   0.10    0.25      1.181771e+01  0.4628045  8.598093e+00
##   0.10    0.30      1.169414e+01  0.4754551  8.543814e+00
##   0.10    0.35      1.162966e+01  0.4847180  8.542738e+00
##   0.10    0.40      1.163123e+01  0.4901694  8.564810e+00
##   0.10    0.45      1.168191e+01  0.4926407  8.625228e+00
##   0.10    0.50      1.177558e+01  0.4923609  8.702043e+00
##   0.10    0.55      1.189541e+01  0.4899572  8.787164e+00
##   0.10    0.60      1.202476e+01  0.4869813  8.884230e+00
##   0.10    0.65      1.215014e+01  0.4841867  8.975242e+00
##   0.10    0.70      1.227417e+01  0.4811240  9.065488e+00
##   0.10    0.75      1.239370e+01  0.4780664  9.156581e+00
##   0.10    0.80      1.250599e+01  0.4749700  9.238030e+00
##   0.10    0.85      1.261585e+01  0.4717016  9.319321e+00
##   0.10    0.90      1.272355e+01  0.4683362  9.401281e+00
##   0.10    0.95      1.282767e+01  0.4651530  9.479352e+00
##   0.10    1.00      1.292751e+01  0.4619990  9.552861e+00
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were fraction = 0.35 and lambda = 0.1.
```



```
## [1] "The final value used for the model has 0.1 as lambda value and 0.35 as frac
tion value"
```

```
## [1] "The RMSE value for elastic net model on the train set 11.6297"
```

```
## [1] "The R^2 value for elastic net model on the train set 0.4847"
```

```
## [1] "The RMSE value for elastic net model on the test set 11.1501"
```

```
## [1] "The R^2 value for elastic net model on the test set 0.4271"
```

Thus the above plot shows that the best enet model has the parameter of fraction value as 0.35 and lambda as 0.1 that is choosed from the train set using Elastic net. And followed by we have the model’s tuning parameter, RMSE and R square value of the train and test sets.

## Overall Results

After doing all five models the below table shows the Best parameter, RMSE and R square value of the Training and test set.

| Model | Parameter | Training RMSE | Training R Squared | Testing RMSE | Testing R Squared |
|---|---|---|---|---|---|
| Linear | 388 predictors | 47.2918 | 0.1072 | 24.0788 | 0.0325 |
| PLS | 7 components | 11.9295 | 0.4759 | 10.8979 | 0.4457 |
| Ridge | Lambda as 0.1 | 12.9275 | 0.462 | 12.1947 | 0.385 |
| Lasso | Fraction as 0.2 | 1093454.0597 | 0.3858 | 11.4899 | 0.3907 |
| Elastic Net | lambda as 0.1 & Fraction as 0.35 | 11.6297 | 0.4847 | 11.1501 | 0.4271 |

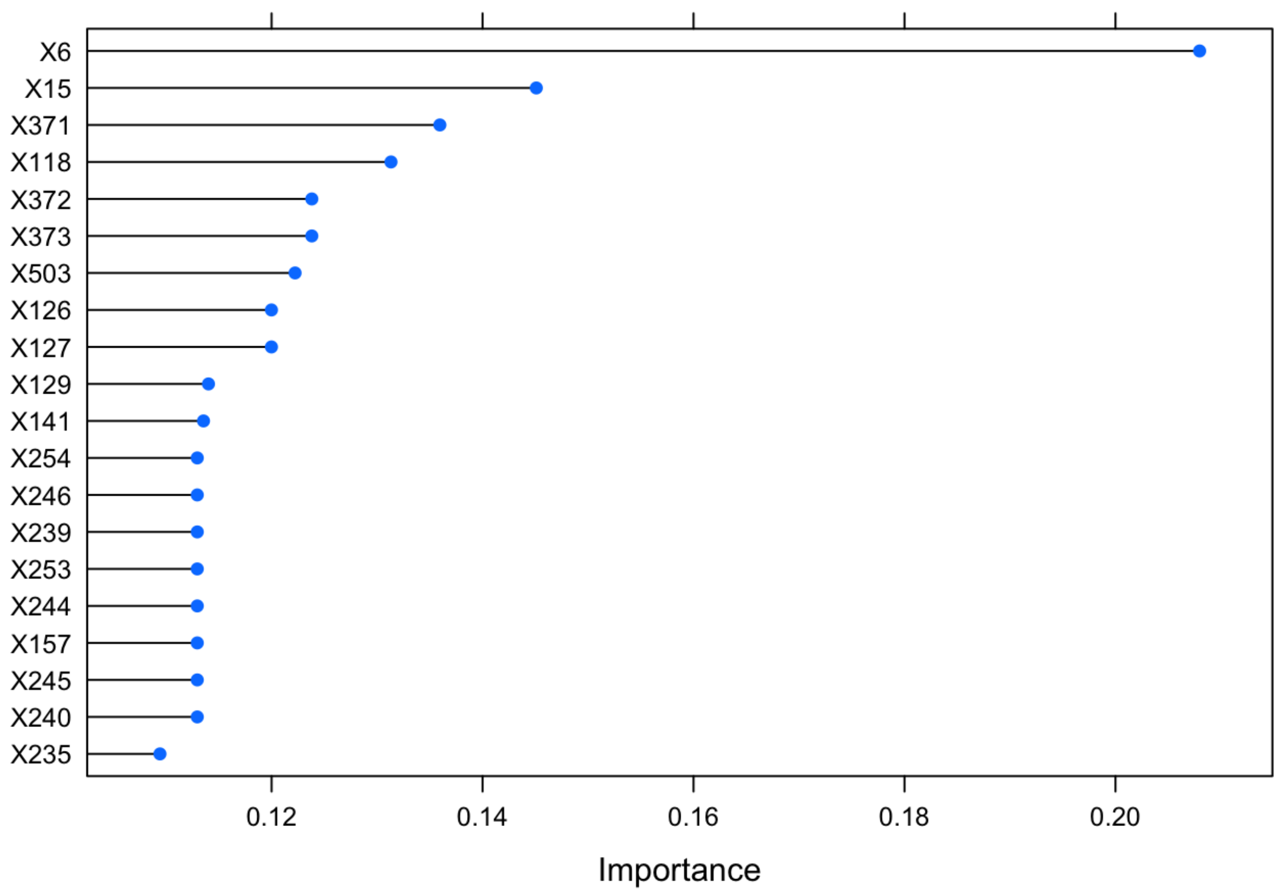
Looking at the table we can say that, PLS is the best for this dataset. Thus the model generated using other models doesn’t have better predictive performance than PLS for this test and train split with seed as 1.

## Section (f)

By looking at the above table and expecially the testing R square value, I won’t recommend any model to replace the permeability laboratory experiment because the R square values are not high or close to 1.

## Section (g)

In order to look at the important predictor in our list I have used varImp() to get the importance of each and every predictors and sorted it to identify the top 10 predictors in it. Below the list of top 10 important predictors.



From the graph we can see that the first predictor X6 with ~0.21 importance is kind of dominating the list.

*** End of Solution ***

## Appendix - Coding

# Question 1

## Section (a)

### Subset of predictors and target variable

/# Load the data

data(tecator)

absorp_df <- as.data.frame(absorp)

endpoints_df <- as.data.frame(endpoints)

colnames(endpoints_df) = c(“mositure”, “fat”,“protein”)

/# Verify the data is loaded or not

head(absorp_df[1:5])

head(endpoints_df)

## Section (b)

/# Finding the Principal Components

pc <- prcomp(absorp)

/# Looking at the variance and finding the total variance

vars <- (pc$sdev^2/sum(pc$sdev^2)) * 100

/# Plot the variance for the components

plot(vars, xlim = c(0, length(vars)), type = ‘b’, pch = 16, xlab=‘number of components’, ylab=‘percent of total variation’,

```
main = "Principle Component Analysis")
```

/# Top 5 principle components

print(“Top 5 PCs”)

vars[1:5]

## Section (c)

/# Setting the seed for reproduciablity

set.seed(1)

/# Performing data spliting

cv_index <- createDataPartition(endpoints[, 3], p = 0.8, list = FALSE)

absorpTrain <- absorp_df[cv_index,]

absorpTest <- absorp_df[-cv_index,]

fatTrain <- endpoints_df[cv_index, 2]

fatTest <- endpoints_df[-cv_index, 2]

/# Setting up the control parameter

ctrl <- trainControl(method = “LGOCV”, repeats = 5)

## Linear model

/# Using coleration to find the correlated predictors

corThresh <- .9

tooHigh <- findCorrelation(cor(absorpTrain), corThresh)

corrPred <- names(absorpTrain)[-tooHigh]

print(“Not correlated predictors in the data is”)

corrPred

/# Filter the only applicable data

absorpTrainFiltered <- data.frame(absorpTrain[, -tooHigh])

absorpTestFiltered <- data.frame(absorpTest[, -tooHigh])

colnames(absorpTestFiltered) <- “absorpTrain….tooHigh.”

set.seed(1)

/# Creating Linear model

lm_model <- train(absorpTrainFiltered, fatTrain, method = “lm”, trControl = ctrl)

/# Print the model

lm_model

/# Finding the root mean square and R^2

lm_pred <- predict(lm_model, absorpTestFiltered)

lm_rmse <-RMSE(fatTest, lm_pred)

lm_R2 <- cor(lm_pred, fatTest, method = “pearson”) ^ 2

/# Make the prediction plot

plot(lm_pred, fatTest, xlab = “Predicted”, ylab = “Observed”, main = “Predicted Vs Observed”, pch = 16, cex = 1.3, col = “blue”)

abline(a = as.numeric(lm_model*finalModel*coefficients[1]),

```
  b = as.numeric(lm_model$finalModel$coefficients[2]), col = "red")
```

/# Testset R Squared and rmse

print(“Linear model has used only first predictor V100 in the model according to correlation cut off 0.9”)

print(paste(“The RMSE value for linear model on the train set”, round(lm_model*results*RMSE, 4)))

print(paste(“The R^2 value for linear model on the train set”, round(lm_model*results*Rsquared, 4)))

print(paste(“The RMSE value for linear model on the test set”, round(lm_rmse, 4)))

print(paste(“The R^2 value for linear model on the test set”, round(lm_R2, 4)))

## Partial Least Square

set.seed(1)

/# Create the model

pls_model <- train(x = absorpTrain, y = fatTrain, method = “pls”, trControl = ctrl, preProcess = c(“center”, “scale”), tuneLength = 20)

/# Print the model

pls_model

/# Plot the results

plot(pls_model, type = c(“p”, “g”), xlab = “Components”, ylab = “RMSE”)

/# Make prediction on the test set

pls_pred <- predict(pls_model, absorpTest, ncomp = as.numeric(pls_model$bestTune))

/# Finding the root mean square and R^2

pls_rmse <-RMSE(fatTest, pls_pred)

pls_R2 <- cor(pls_pred, fatTest, method = “pearson”) ^ 2

/# Print the results and tune parameter

print(paste(“The final value used for the model has”, as.numeric(pls_model$bestTune), “components”))

print(paste(“The RMSE value for pls model on the train set”, round(pls_model$results[as.\ numeric(pls_model$ bestTune), ]$RMSE, 4)))

print(paste(“The R^2 value for pls model on the train set”, round(pls_model$results[as.\ numeric(pls_model$ bestTune), ]$Rsquared, 4)))

print(paste(“The RMSE value for pls model on the test set”, round(pls_rmse, 4)))

print(paste(“The R^2 value for pls model on the test set”, round(pls_R2, 4)))

## Ridge Regression

/# Create the grid for the ridge model

ridge_grid <- data.frame(.lambda = seq(0, 1, length = 20))

set.seed(1)

/# Create the model

ridge_model <- train(x = absorpTrain, y = fatTrain, method = “ridge”, trControl = ctrl, preProcess = c(“center”, “scale”), tuneGrid = ridge_grid)

/# Print the model

ridge_model

/# Plot the results

plot(ridge_model, type = c(“p”, “g”), xlab = “Lambda”, ylab = “RMSE”)

/# Make prediction on the test set

ridge_pred <- predict(ridge_model, as.matrix(absorpTest), s = as.numeric(ridge_model$bestTune))

/# Finding the root mean square and R^2

ridge_rmse <-RMSE(fatTest, ridge_pred)

ridge_R2 <- cor(ridge_pred, fatTest, method = “pearson”) ^ 2

/# Print the results and tune parameter

print(paste(“The final value used for the model has”, as.numeric(ridge_model$bestTune), “as lambda value”))

best_param_value <- ridge_model$results[ridge_model$results$lambda == as.\ numeric(ridge_model$bestTune), ]

print(paste(“The RMSE value for ridge model on the train set”, round(best_param_value$RMSE, 4)))

print(paste(“The R^2 value for rigde model on the train set”, round(best_param_value$Rsquared, 4)))

print(paste(“The RMSE value for ridge model on the test set”, round(ridge_rmse, 4)))

print(paste(“The R^2 value for rigde model on the test set”, round(ridge_R2, 4)))

/#### Lasso Regression

/# Create the grid for the ridge model

```
lasso_grid <- data.frame(.lambda =0, .fraction = seq(0, 1, length = 20))

set.seed(1)

/# Create the model

lasso_model <- train(x = absorpTrain, y = fatTrain, method = “enet”, trControl = ctrl, preProcess = c(“center”,
“scale”), tuneGrid = lasso_grid)

/# Print the model

lasso_model

/# Plot the results

plot(lasso_model, type = c(“p”, “g”), xlab = “Fraction”, ylab = “RMSE”)

/# Make prediction on the test set

lasso_pred <- predict(lasso_model, as.matrix(absorpTest), s = as.numeric(lasso_model$bestTune))

/# Finding the root mean square and R^2

lasso_rmse <-RMSE(fatTest, lasso_pred)

lasso_R2 <- cor(lasso_pred, fatTest, method = “pearson”) ^ 2

/# Print the results and tune parameter

print(paste(“The final value used for the model has”, as.numeric(lasso_model$bestTune[1]), “as fraction
value”))
```

best_param_value <- lasso_model$results[lasso_model$results$fraction == as. numeric(lasso_model$bestTune[1]), ]

```
print(paste(“The RMSE value for lasso model on the train set”, round(best_param_value$RMSE, 4)))

print(paste(“The R^2 value for lasso model on the train set”, round(best_param_value$Rsquared, 4)))

print(paste(“The RMSE value for lasso model on the test set”, round(lasso_rmse, 4)))

print(paste(“The R^2 value for lasso model on the test set”, round(lasso_R2, 4)))

/#### Elastic Net

/# Develop the grid

enetGrid <- expand.grid(.lambda = c(0, 0.01, .1), .fraction = seq(.05, 1, length = 20))

set.seed(1)

/# Train the Enet model

enet_model <- train(x = absorpTrain, y = fatTrain, method = “enet”, tuneGrid = enetGrid, trControl = ctrl,
preProc = c(“center”, “scale”))

/# Print the model

print(enet_model)

/# Plot the paramter to see the best parameter

plot(enet_model, type = c(“p”, “g”), xlab = “Fraction”, ylab = “RMSE”)

/# Make prediction on the test set

enet_pred <- predict(enet_model, as.matrix(absorpTest), s = as.numeric(enet_model$bestTune), mode =
```

“fraction”)

/# Finding the root mean square and R^2

enet_rmse <-RMSE(fatTest, enet_pred)

enet_R2 <- cor(enet_pred, fatTest, method = “pearson”) ^ 2

/# Print the results and tune parameter

print(paste(“The final value used for the model has”, as.numeric(enet_model
$bestTune[2]),\ "\ aslambdavalueand\ "\ ,\ enet_{m}odel$bestTune[1], “as fraction value”))

best_param_value <- enet_model$results[enet_{m}odel$results$fraction == as.\ numeric(enet_{m}odel$bestTune)[1] & enet_model$results$lambda == as.numeric(enet_model$bestTune)[2], ]

print(paste(“The RMSE value for elastic net model on the train set”, round(best_param_value$RMSE, 4)))

print(paste(“The R^2 value for elastic net model on the train set”, round(best_param_value$Rsquared, 4)))

print(paste(“The RMSE value for elastic net model on the test set”, round(enet_rmse, 4)))

print(paste(“The R^2 value for elastic net model on the test set”, round(enet_R2, 4)))

## Question 2

### Section (a)

data(permeability)

fingerprints_df <- as.data.frame(fingerprints)

head(fingerprints_df[, 1:5])

permeability[1:5]

### Section (b)

fingerprints_filtered_df <- fingerprints_df[, -nearZeroVar(fingerprints_df)]

print(paste(“Number of predictors left out for modeling is”, dim(fingerprints_filtered_df)[2]))

### Section (c)

/# Setting the seed for reproduciablity

set.seed(1)

/# Performing data spliting

cv_index <- createDataPartition(permeability, p = 0.8, list = FALSE)

fingerprintsTrain <- fingerprints_filtered_df[cv_index,]

fingerprintsTest <- fingerprints_filtered_df[-cv_index,]

permeabilityTrain <- permeability[cv_index]

permeabilityTest <- permeability[-cv_index]

/# Setting up the control parameter

ctrl <- trainControl(method = “LGOCV”)

set.seed(1)

/# Create the model

pls_model <- train(x = fingerprintsTrain, y = permeabilityTrain, method = “pls”, trControl = ctrl, preProcess = c(“center”, “scale”), tuneLength = 20)

/# Print the model

pls_model

/# Plot the results

plot(pls_model, type = c(“p”, “g”), xlab = “Components”, ylab = “RMSE”)

/# Make prediction on the test set

pls_pred <- predict(pls_model, fingerprintsTest, ncomp = as.numeric(pls_model$bestTune))

/# Finding the root mean square and R^2

pls_rmse <-RMSE(permeabilityTest, pls_pred)

pls_R2 <- cor(pls_pred, permeabilityTest, method = “pearson”) ^ 2

/# Print the results and tune parameter

print(paste(“The final value used for the model has”, as.numeric(pls_model$bestTune), “components”))

print(paste(“The RMSE value for pls model on the train set”, round(pls_model$results[as.\ numeric(pls_m odel$
bestTune), ]$RMSE, 4)))

print(paste(“The R^2 value for pls model on the train set”, round(pls_model$results[as.\ numeric(pls_m odel$
bestTune), ]$Rsquared, 4)))

print(paste(“The RMSE value for pls model on the test set”, round(pls_rmse, 4)))

print(paste(“The R^2 value for pls model on the test set”, round(pls_R2, 4)))

## Section (d)

pls_pred

## Section (e)

## Linear model

set.seed(1)

/# Creating Linear model

lm_model <- train(fingerprintsTrain, permeabilityTrain, method = “lm”, trControl = ctrl)

/# Finding the root mean square and R^2

lm_pred <- predict(lm_model, fingerprintsTest)

lm_rmse <-RMSE(lm_pred, permeabilityTest)

lm_R2 <- cor(lm_pred, permeabilityTest, method=“pearson”) ^ 2

```
/# Make the prediction plot

plot(lm_pred, permeabilityTest, cex = 1.3, pch = 16, col = “blue”, xlab = “Predicted”, ylab = “Observed”, main
= “Predicted vs Observed”)

/# Testset R Squared and rmse

print(paste(“The RMSE value for linear model on the train set”, round(lm_modelresultsRMSE, 4)))

print(paste(“The R^2 value for linear model on the train set”, round(lm_modelresultsRsquared, 4)))

print(paste(“The RMSE value for linear model on the test set”, round(lm_rmse, 4)))

print(paste(“The R^2 value for linear model on the test set”, round(lm_R2, 4)))
```

## Ridge Regression

```
/# Create the grid for the ridge model

ridge_grid <- data.frame(.lambda = seq(0, .1, length = 20))

set.seed(1)

/# Create the model

ridge_model <- train(x = fingerprintsTrain, y = permeabilityTrain, method = “ridge”, trControl = ctrl,
preProcess = c(“center”, “scale”), tuneGrid = ridge_grid)

/# Print the model

ridge_model

/# Plot the results

plot(ridge_model, type = c(“p”, “g”), xlab = “Lambda”, ylab = “RMSE”)

/# Make prediction on the test set

ridge_pred <- predict(ridge_model, as.matrix(fingerprintsTest), s = as.numeric(ridge_model$bestTune))

/# Finding the root mean square and R^2

ridge_rmse <-RMSE(permeabilityTest, ridge_pred)

ridge_R2 <- defaultSummary(data.frame( obs = permeabilityTest, pred = ridge_pred))[2]

/# Print the results and tune parameter

print(paste(“The final value used for the model has”, as.numeric(ridge_model$bestTune), “as lambda
value”))

best_param_value <- ridge_modelresults[ridge_modelresultslambda == as. numeric(ridge_modelbestTune), ]

print(paste(“The RMSE value for ridge model on the train set”, round(best_param_value$RMSE, 4)))

print(paste(“The R^2 value for rigde model on the train set”, round(best_param_value$Rsquared, 4)))

print(paste(“The RMSE value for ridge model on the test set”, round(ridge_rmse, 4)))

print(paste(“The R^2 value for rigde model on the test set”, round(ridge_R2, 4)))
```

## Lasso Regression

```
/# Create the grid for the ridge model
```

lasso_grid <- expand.grid(.lambda = 0, .fraction = seq(0.05, 1, length = 20))

set.seed(1)

/# Create the model

lasso_model <- train(x = fingerprintsTrain, y = permeabilityTrain, method = “enet”, trControl = ctrl, preProcess = c(“center”, “scale”), tuneGrid = lasso_grid)

/# Print the model

lasso_model

/# Plot the results

plot(lasso_model, type = c(“p”, “g”), xlab = “Fraction”, ylab = “RMSE”)

/# Make prediction on the test set

lasso_pred <- predict(lasso_model, as.matrix(fingerprintsTest), s = as.numeric(lasso_model$bestTune)[1], mode = “fraction”)

/# Finding the root mean square and R^2

lasso_rmse <- RMSE(permeabilityTest, lasso_pred)

lasso_R2 <- defaultSummary(data.frame(obs = permeabilityTest, pred = lasso_pred))[2]

/# Print the results and tune parameter

print(paste(“The final value used for the model has”, as.numeric(lasso_model$bestTune[1]), “as fraction value”))

best_param_value <- lasso_model$results[lasso_{m}odel$results$fraction == as.numeric(lasso_{m}odel$bestTune[1]), ]

print(paste(“The RMSE value for lasso model on the train set”, round(best_param_value$RMSE, 4)))

print(paste(“The R^2 value for lasso model on the train set”, round(best_param_value$Rsquared, 4)))

print(paste(“The RMSE value for lasso model on the test set”, round(lasso_rmse, 4)))

print(paste(“The R^2 value for lasso model on the test set”, round(lasso_R2, 4)))

## Elastic Net

/# Develop the grid

enetGrid <- expand.grid(.lambda = c(0, 0.01, .1), .fraction = seq(0.05, 1, length = 20))

set.seed(1)

/# Train the Enet model

enet_model <- train(x = fingerprintsTrain, y = permeabilityTrain, method = “enet”, tuneGrid = enetGrid, trControl = ctrl, preProc = c(“center”, “scale”))

/# Print the model

enet_model

/# Plot the paramter to see the best parameter

plot(enet_model, type = c(“p”, “g”), xlab = “Fraction”, ylab = “RMSE”)

/# Make prediction on the test set

enet_pred <- predict(enet_model, as.matrix(fingerprintsTest), s = as.numeric(enet_model$bestTune), mode = “fraction”)

/# Finding the root mean square and R^2

enet_rmse <-RMSE(permeabilityTest, enet_pred)

enet_R2 <- defaultSummary(data.frame(obs = permeabilityTest, pred = enet_pred))[2]

/# Print the results and tune parameter

print(paste(“The final value used for the model has”, as.numeric(enet_model
$bestTune[2]), " aslambdavalueand " , enet_{m}odel$bestTune[1], “as fraction value”))

best_param_value <- enet_model$results[enet_{m}odel$results$fraction == as.numeric(enet_{m}odel$bestTune)[1] & enet_model$results$lambda == as.numeric(enet_model$bestTune)[2], ]

print(paste(“The RMSE value for elastic net model on the train set”, round(best_param_value$RMSE, 4)))

print(paste(“The R^2 value for elastic net model on the train set”, round(best_param_value$Rsquared, 4)))

print(paste(“The RMSE value for elastic net model on the test set”, round(enet_rmse, 4)))

print(paste(“The R^2 value for elastic net model on the test set”, round(enet_R2, 4)))

## Section (g)

/# Getting the important predictors

important <- varImp(pls_model$finalModel, useModel=pls, scale = FALSE)

important_order <- order(important, decreasing = TRUE)

importance_df <- as.data.frame(list(ColumnName = names(fingerprints_filtered_df[, important_order]), Importance = important$Overall[important_order]))

top_10 <- importance_df[1:10, ]

top_10

/# Plot it

plot(top_10$Importance, type=“l”)

Processing math: 100%

## ignemnt