Project 1

Rohan.P| Mithil.G| Shiva Sai.V| Koushik.K

2023-04-22

Health Data Prediction

Loading data from the provided file in data.

```
head(data)
```

```
## # A tibble: 6 × 5
##      X1    X2    X3    X4    X5
##   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  8       78   284  9.10   109
## 2  9.30    68   433  8.70   144
## 3  7.5     70   739  7.20   113
## 4  8.90    96  1792  8.90    97
## 5 10.2     74   477  8.30   206
## 6  8.30   111   362 10.9    124
```

```
summary(data)
```

```
##        X1               X2              X3               X4        
##  Min.   : 3.600   Min.   : 60.0   Min.   : 190.0   Min.   : 7.200  
##  1st Qu.: 8.300   1st Qu.: 82.0   1st Qu.: 353.0   1st Qu.: 8.800  
##  Median : 9.400   Median :114.0   Median : 525.0   Median : 9.500  
##  Mean   : 9.306   Mean   :116.1   Mean   : 589.8   Mean   : 9.436  
##  3rd Qu.:10.300   3rd Qu.:134.0   3rd Qu.: 686.0   3rd Qu.:10.300  
##  Max.   :12.800   Max.   :238.0   Max.   :1792.0   Max.   :13.000  
##        X5       
##  Min.   : 35.0  
##  1st Qu.: 80.0  
##  Median :103.0  
##  Mean   :110.6  
##  3rd Qu.:129.0  
##  Max.   :292.0
```

Checking for missing data in every column

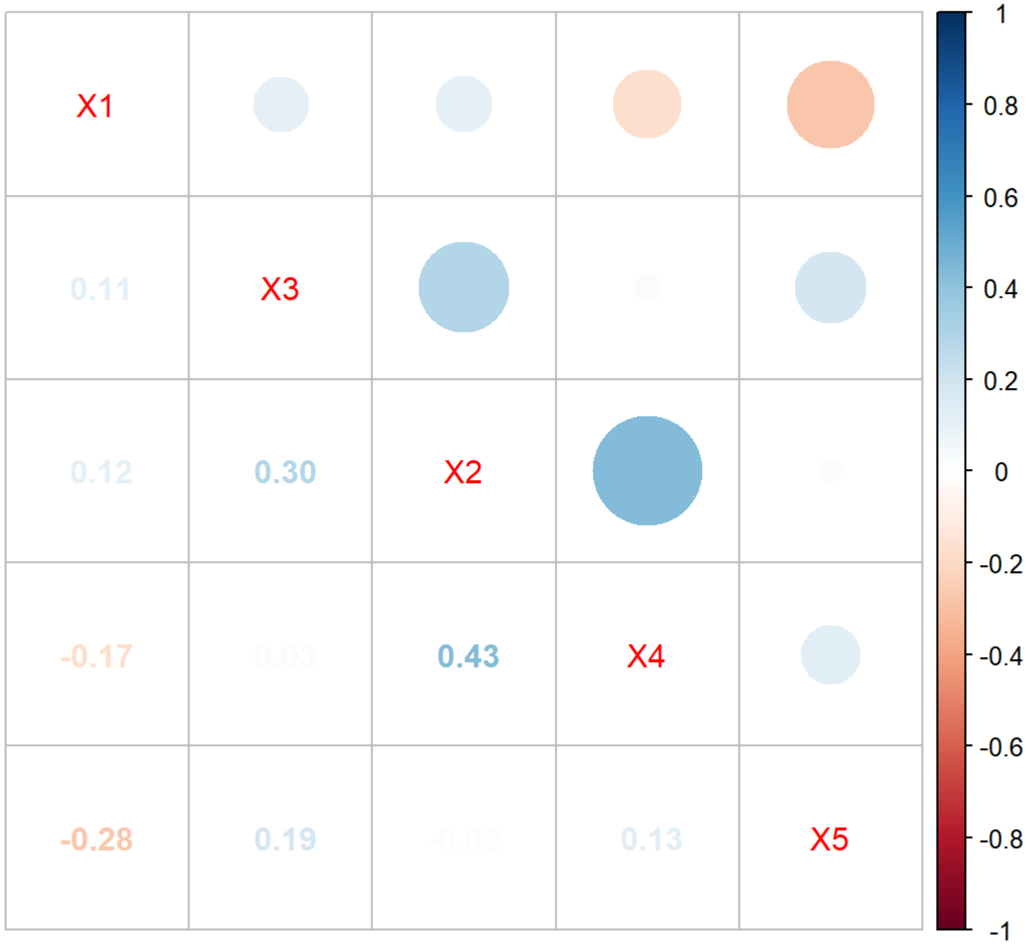
```
any(is.na(data))
```

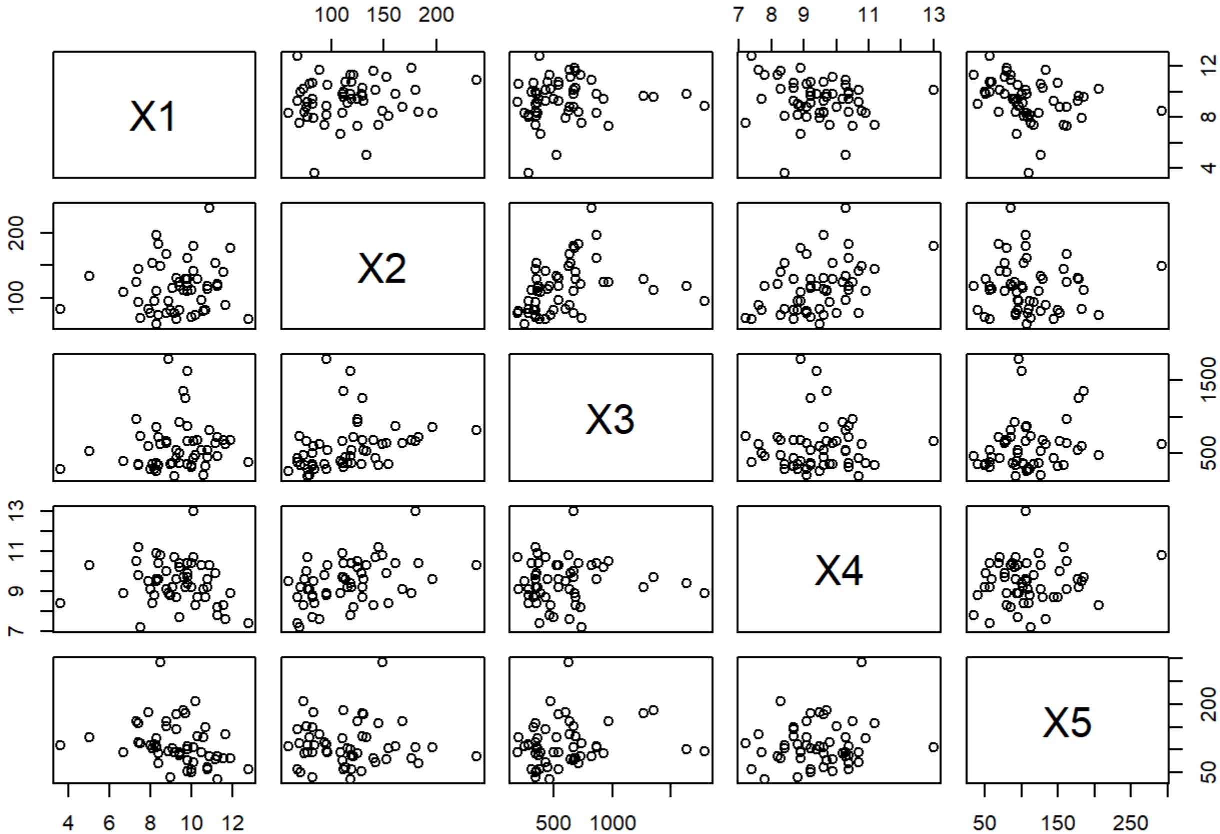
```
## [1] FALSE
```

There is no missing data in the provided file.

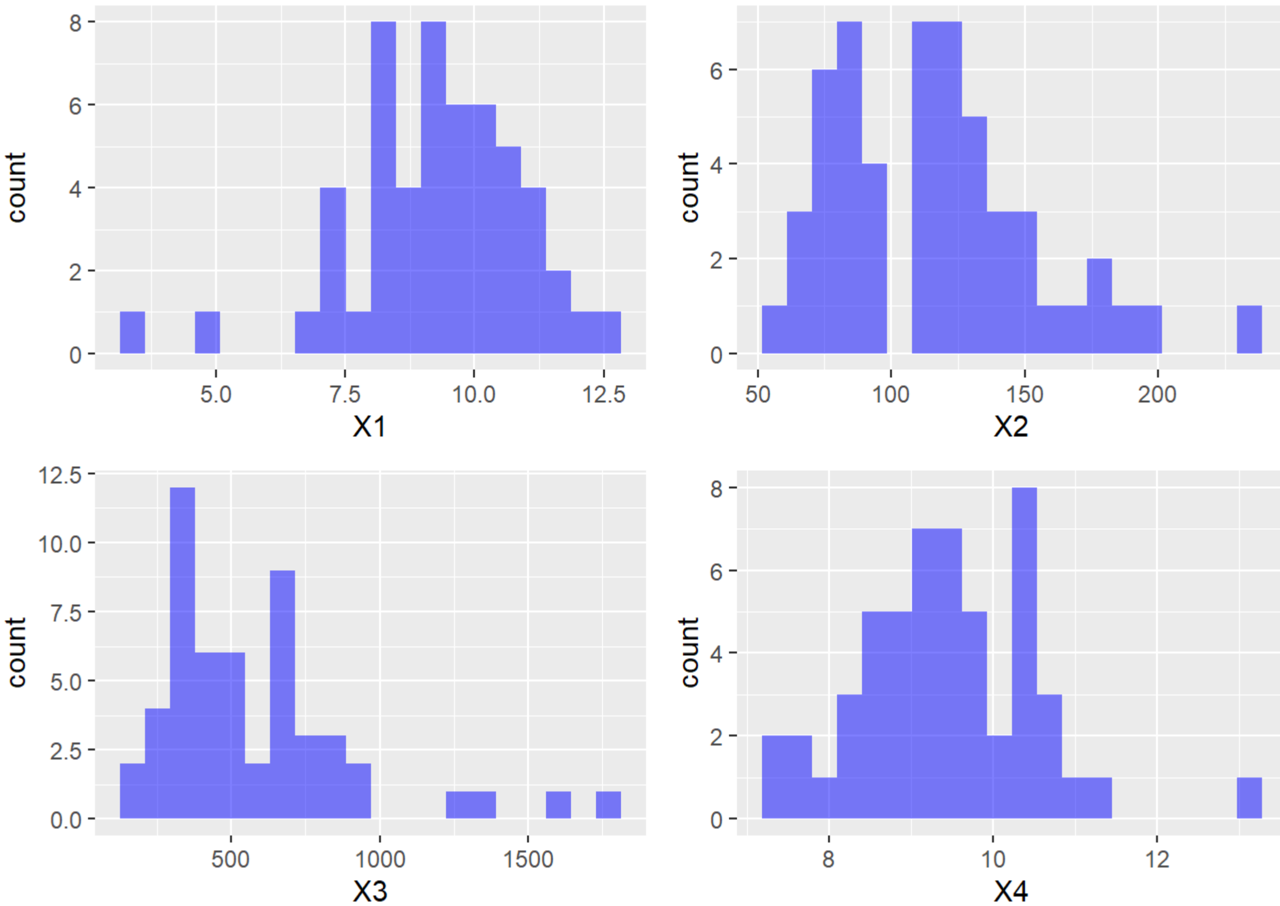
Plotting the data: respose vs every predictors and analysing using histogram

Plotting correlation plot and a column vs every other column

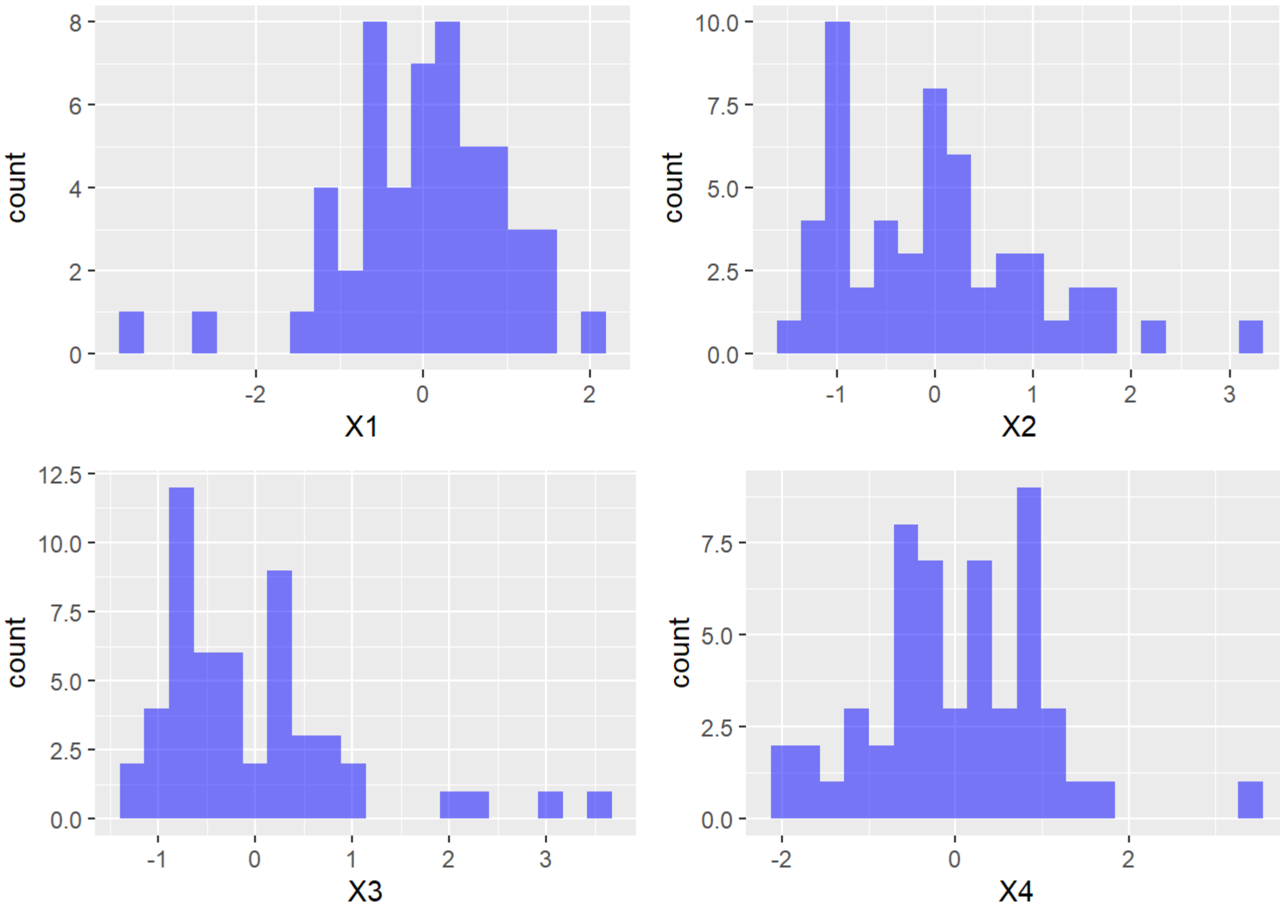



X1
X2
X3
X4
X5
100 150 200
7 8 9 11 13
12 8 4
200 100
1500 500
13 11 9 7
200 50
4 6 8 10 12
500 1000
50 150 250

Without scaling



After scaling the data



Splitting the data randomly in the ratio 0.8

```
rand_sample <- sample.split(X_std$X1, SplitRatio = 0.8)
tr_data = subset(data, rand_sample == TRUE)
test_data = subset(data, rand_sample == FALSE)
```

Linear Model

Fitting the linear model to our data: Multiple Linear Regression

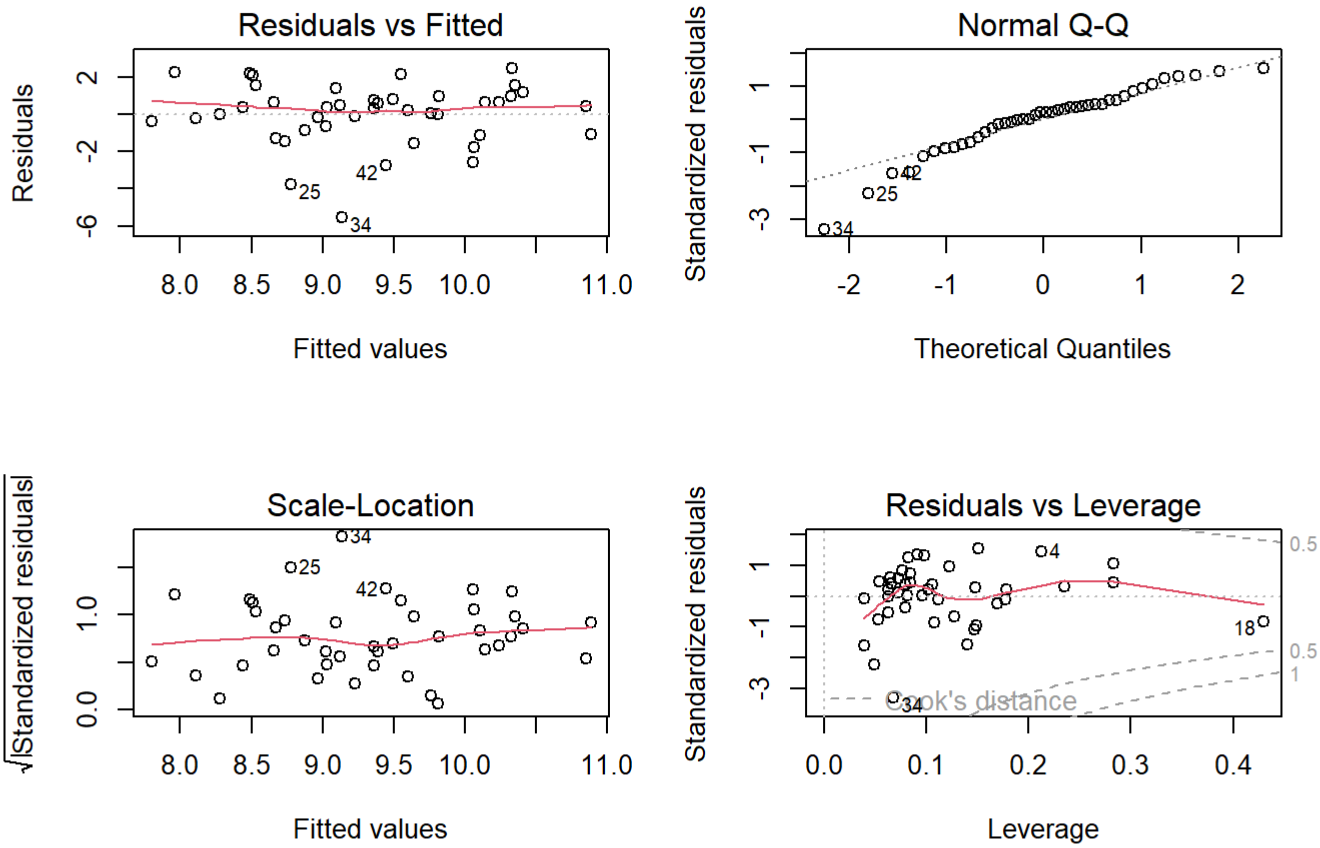
```
model <- lm(X1 ~ ., data = tr_data)
summary(model)
```

```
##
## Call:
## lm(formula = X1 ~ ., data = tr_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.5384 -0.8124  0.3500  0.9346  2.4708
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.941468   2.376118   5.446 3.53e-06 ***
## X2           0.004479   0.008806   0.509   0.6140
## X3           0.001347   0.001028   1.311   0.1980
## X4          -0.349895   0.274152  -1.276   0.2098
## X5          -0.014802   0.007185  -2.060   0.0465 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.731 on 37 degrees of freedom
## Multiple R-squared:  0.1827, Adjusted R-squared:  0.09432
## F-statistic: 2.067 on 4 and 37 DF,  p-value: 0.1049
```

```
res <- residuals(model)
res_df <- as.data.frame(res)
head(res)
```

```
##           1           2           3           4           5           6
## -0.87604357  0.64610434 -2.55888922  2.23772678  0.02287939  0.35837769
```

```
par(mfrow=c(2,2))
plot(model)
```




```
par(mfrow=c(1,1))
```

```
#train_pred
tr_pred = predict(model,tr_data)
tr_results = cbind(tr_data$X1,tr_pred)
colnames(tr_results) = c("actual","predicted")
tr_results <- as.data.frame(tr_results)
tr_mse <- mean((tr_results$actual - tr_results$predicted)^2)
tr_ssr <- sum((tr_results$actual - tr_results$predicted)^2)
tr_sst <- sum((mean(tr_data$X1) - tr_results$predicted)^2)
tr_R2 <- 1 -(tr_ssr/tr_sst)
```

```
##   actual predicted
## 1    8.9 11.236238
## 2    8.5  6.358000
## 3    8.3  8.633877
## 4    8.2  9.032037
## 5    9.3  8.282284
## 6   11.2  9.881430
```

- **Linear Model**
  - Train MSE : 2.6407915
  - Test MSE : 1.5890087

- R sq. : -0.0630718

Polynomial Regression: With single predictor

Lets try each predictor with polynomial degree.

```
x2_mod <- lm(X1 ~ poly(X2,degree = 2), data = tr_data)
summary(x2_mod)
```

```
##
## Call:
## lm(formula = X1 ~ poly(X2, degree = 2), data = tr_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.6050 -1.1417  0.3521  1.2131  3.5753
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)             9.3429     0.2851  32.765   <2e-16 ***
## poly(X2, degree = 2)1   1.3918     1.8479   0.753    0.456
## poly(X2, degree = 2)2   0.7650     1.8479   0.414    0.681
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.848 on 39 degrees of freedom
## Multiple R-squared:  0.01859,    Adjusted R-squared:  -0.03174
## F-statistic: 0.3693 on 2 and 39 DF,  p-value: 0.6936
```

None of the degrees is significant and Adjusted R^2 is -3.1740487%

```
x3_mod <- lm(X1 ~ poly(X3,degree = 2), data = tr_data)
summary(x3_mod)
```

```
##
## Call:
## lm(formula = X1 ~ poly(X3, degree = 2), data = tr_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.2064 -0.8630  0.1094  1.0951  3.7208
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)             9.3429     0.2817  33.167   <2e-16 ***
## poly(X3, degree = 2)1   1.6302     1.8256   0.893    0.377
## poly(X3, degree = 2)2  -1.7519     1.8256  -0.960    0.343
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.826 on 39 degrees of freedom
## Multiple R-squared:  0.0422, Adjusted R-squared:  -0.006916
## F-statistic: 0.8592 on 2 and 39 DF,  p-value: 0.4314
```

None of the degrees is significant and adjusted R^2 is -0.6916267%

```
X4_mod <- lm(X1 ~ poly(X4,degree = 2), data = tr_data)
summary(X4_mod)
```

```
##
## Call:
## lm(formula = X1 ~ poly(X4, degree = 2), data = tr_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.0902 -0.7650  0.4661  1.1968  2.5113
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)             9.3429     0.2787  33.518   <2e-16 ***
## poly(X4, degree = 2)1  -2.1398     1.8064  -1.185    0.243
## poly(X4, degree = 2)2   1.9643     1.8064   1.087    0.284
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.806 on 39 degrees of freedom
## Multiple R-squared:  0.06217,    Adjusted R-squared:  0.01408
## F-statistic: 1.293 on 2 and 39 DF,  p-value: 0.286
```

None of the degrees is significant and adjusted R^2 is 1.4081015%

```
x5_mod <- lm(X1 ~ poly(X5,degree = 2), data = tr_data)
summary(x5_mod)
```

```
##
## Call:
## lm(formula = X1 ~ poly(X5, degree = 2), data = tr_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.2986 -0.9227  0.1051  1.0655  3.0463
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)             9.3429     0.2619  35.671   <2e-16 ***
## poly(X5, degree = 2)1  -3.4737     1.6974  -2.046   0.0475 *
## poly(X5, degree = 2)2   3.3567     1.6974   1.978   0.0551 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.697 on 39 degrees of freedom
## Multiple R-squared:  0.172,  Adjusted R-squared:  0.1295
## F-statistic: 4.049 on 2 and 39 DF,  p-value: 0.02524
```

Looking at the P value degree 1 is statistically significant and adjusted R^2 is 12.9487472%

Polynomial Regression with single predictor did not work so well for the data.

Polynomial Regression

Fitting the Polynomial model to our data:

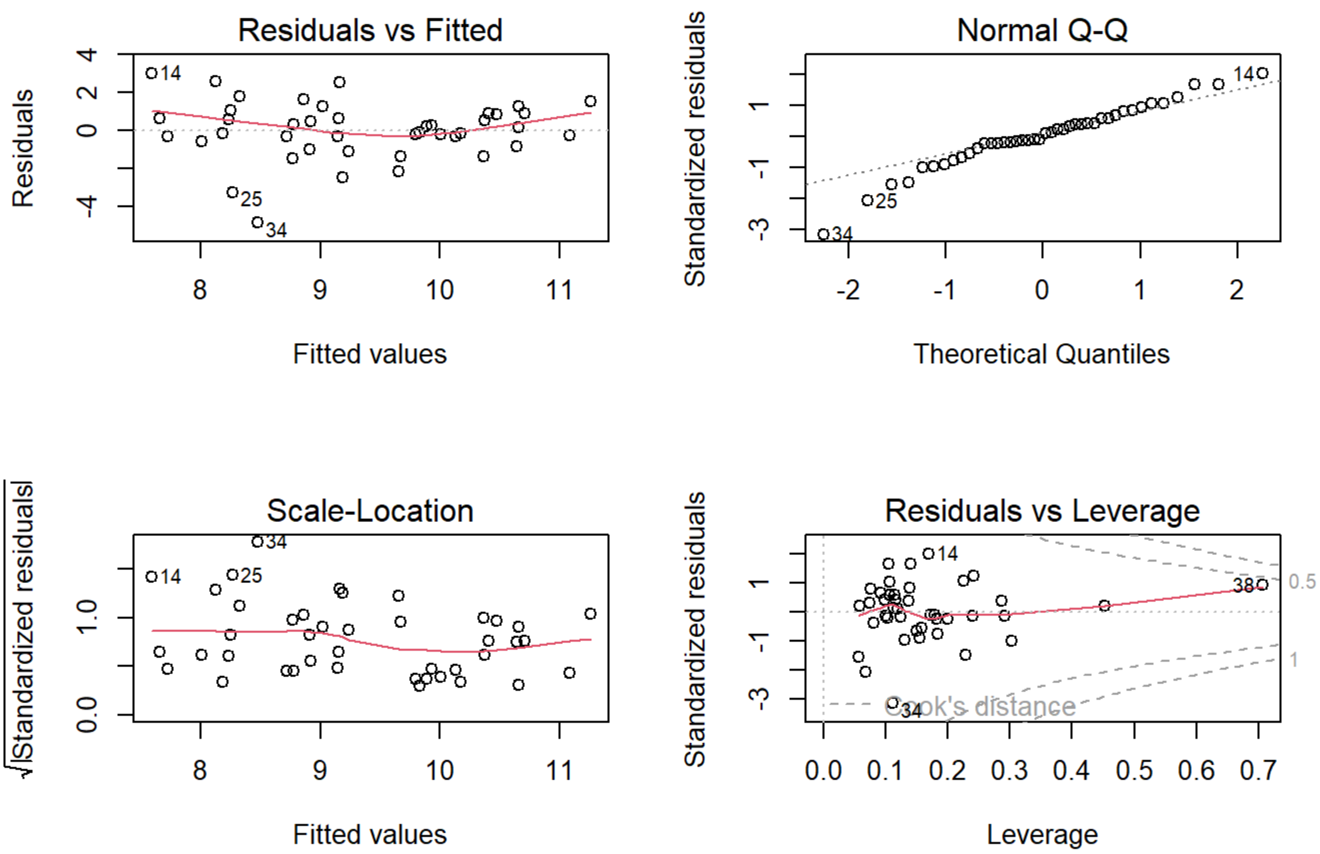
```
x2 <- tr_data$X2
log_X3 <- log(tr_data$X3)
log_X5 <- log(tr_data$X5)
log_X4 <- log(tr_data$X4)
poly_model <- lm(X1 ~ X2 + poly(log_X3,log_X4,degree = 1) + poly(log_X5,degree = 3), data = tr_d
ata)
summary(poly_model)
```

```
##
## Call:
## lm(formula = X1 ~ X2 + poly(log_X3, log_X4, degree = 1) + poly(log_X5,
##     degree = 3), data = tr_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8775 -0.5405  0.0090  0.8767  3.0103
##
## Coefficients:
##                                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)                         8.695409   1.069724   8.129 1.42e-09 ***
## X2                                  0.005517   0.008856   0.623   0.5374
## poly(log_X3, log_X4, degree = 1)1.0  2.469142   1.976271   1.249   0.2198
## poly(log_X3, log_X4, degree = 1)0.1 -2.588650   1.897594  -1.364   0.1812
## poly(log_X5, degree = 3)1          -4.239883   1.798550  -2.357   0.0241 *
## poly(log_X5, degree = 3)2           0.890678   1.720976   0.518   0.6080
## poly(log_X5, degree = 3)3           3.202922   1.662678   1.926   0.0622 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.64 on 35 degrees of freedom
## Multiple R-squared:  0.3065, Adjusted R-squared:  0.1877
## F-statistic: 2.579 on 6 and 35 DF,  p-value: 0.03564
```

```
res_poly <- residuals(poly_model)
res_df_poly <- as.data.frame(res)
head(res_poly)
```

```
##          1          2          3          4          5          6
## -0.1834068  1.0514040 -2.1530006  0.2662736  0.6441266  0.5622782
```

```
par(mfrow=c(2,2))
plot(poly_model)
```




```
par(mfrow=c(1,1))
```

```
#train_pred
tr_pred_poly = predict(poly_model,tr_data)
tr_results_poly = cbind(tr_data$X1,tr_pred_poly)
colnames(tr_results_poly) = c("actual","predicted")
tr_results_poly <- as.data.frame(tr_results_poly)
tr_mse_poly <- mean((tr_results_poly$actual - tr_results_poly$predicted)^2)
tr_ssr_poly <- sum((tr_results_poly$actual - tr_results_poly$predicted)^2)
tr_sst_poly <- sum((mean(tr_data$X1) - tr_results_poly$predicted)^2)
tr_R2_poly <- 1 -(tr_ssr_poly/tr_sst_poly)
```

```
#test_pred
test_pred_poly = predict(poly_model,new_data = test_data)
test_results_poly = cbind(test_data$X1,test_pred_poly)
colnames(test_results_poly) = c("actual","predicted")
test_results_poly <- as.data.frame(test_results_poly)
head(test_results_poly)
```

```
##   actual predicted
## 1    8.9  8.183407
## 2    8.5  8.248596
## 3    8.3  9.653001
## 4    8.2  9.933726
## 5    9.3  7.655874
## 6   11.2  8.237722
```

```
mse_poly<- mean((test_results_poly$actual - test_results_poly$predicted)^2)
ssr_poly <- sum((test_results_poly$actual - test_results_poly$predicted)^2)
sst_poly <- sum((mean(test_data$X1) - test_results_poly$predicted)^2)
R2_poly <- 1 -(ssr_poly/sst_poly)
```

- **Polynomial Model**
  - Train MSE : 2.2406009
  - Test MSE : 1.4660928
  - R sq. : -0.4337759

Random forest

Fitting the Random forest to our data:

```
rf_model = randomForest(X1 ~ .,data = tr_data,mtry=4,importance =TRUE)
rf_model
```

```
## 
## Call:
##  randomForest(formula = X1 ~ ., data = tr_data, mtry = 4, importance = TRUE) 
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
## 
##           Mean of squared residuals: 3.401712
##                     % Var explained: -5.28
```

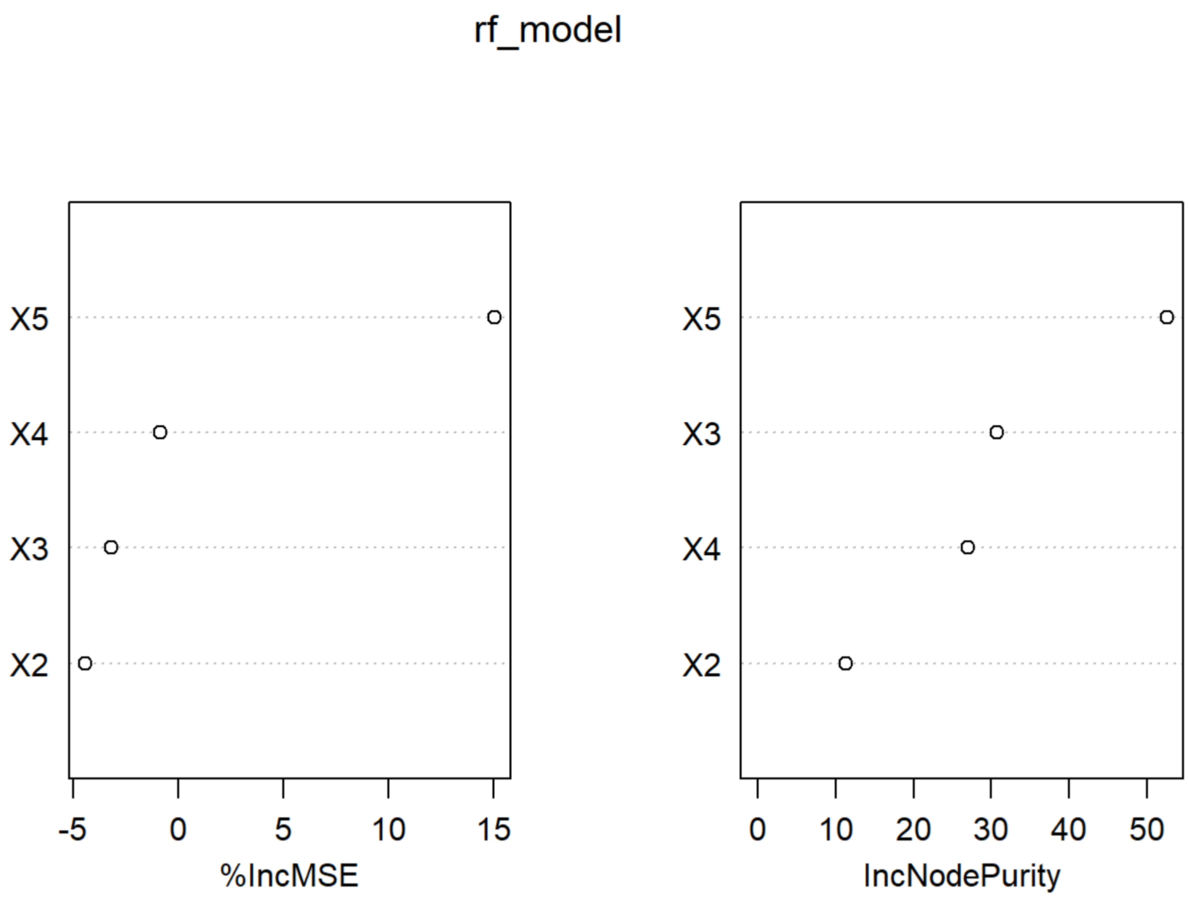
```
summary(rf_model)
```

```
##                 Length Class  Mode
## call              5    -none- call
## type              1    -none- character
## predicted        42    -none- numeric
## mse             500    -none- numeric
## rsq             500    -none- numeric
## oob.times        42    -none- numeric
## importance        8    -none- numeric
## importanceSD      4    -none- numeric
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           11    -none- list
## coefs             0    -none- NULL
## y                42    -none- numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

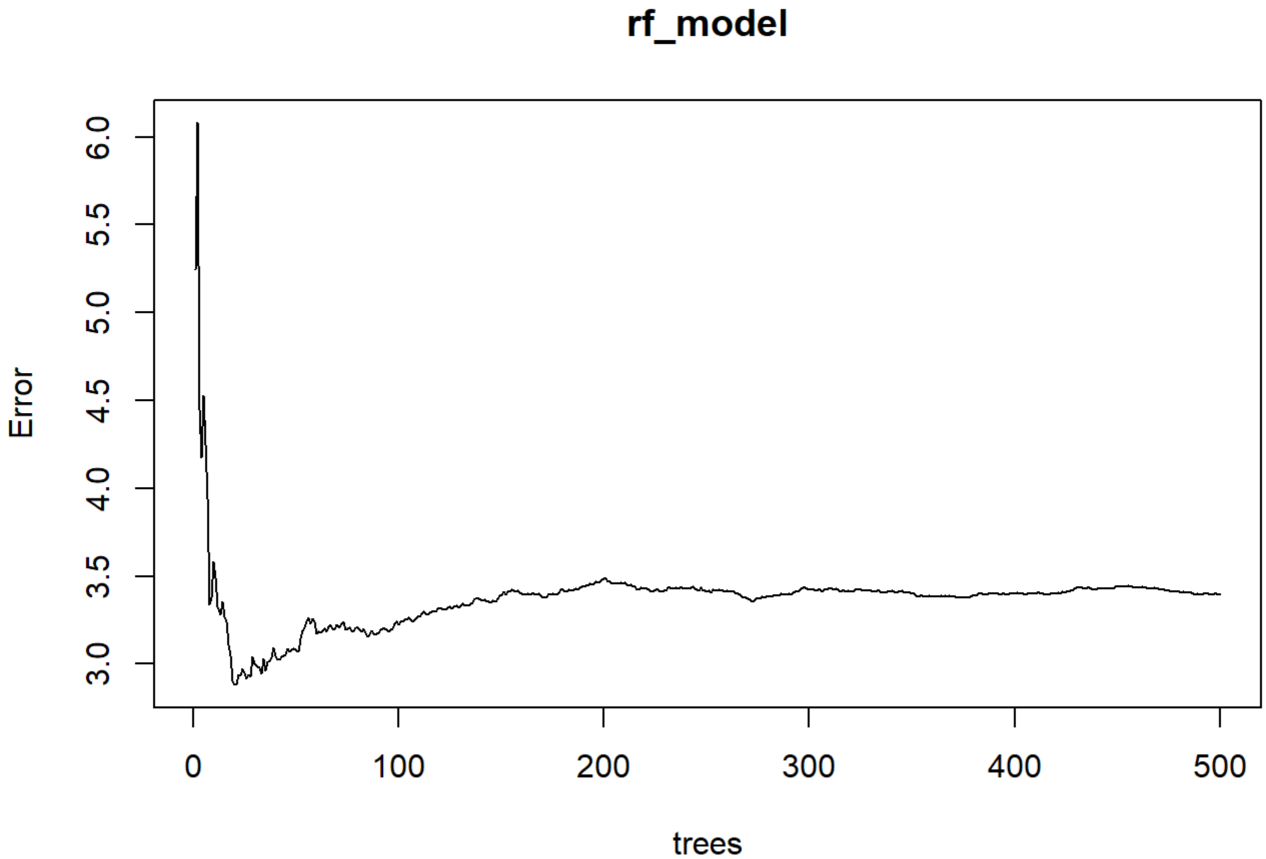
```
importance(rf_model)
```

```
##       %IncMSE IncNodePurity
## X2 -4.4426776      11.29923
## X3 -3.1839532      30.71032
## X4 -0.8673535      27.03891
## X5 15.0334541      52.53463
```

```
varImpPlot(rf_model)
```




```
rf_tr_pred <- predict(rf_model,data=tr_data)
tr_mse_rf <- mean((rf_tr_pred - tr_data$X1)^2)
rf_pred = predict(rf_model,newdata = test_data)
mse_rf <- mean((rf_pred - test_data$X1)^2)
plot(rf_model,data=tr_data)
```

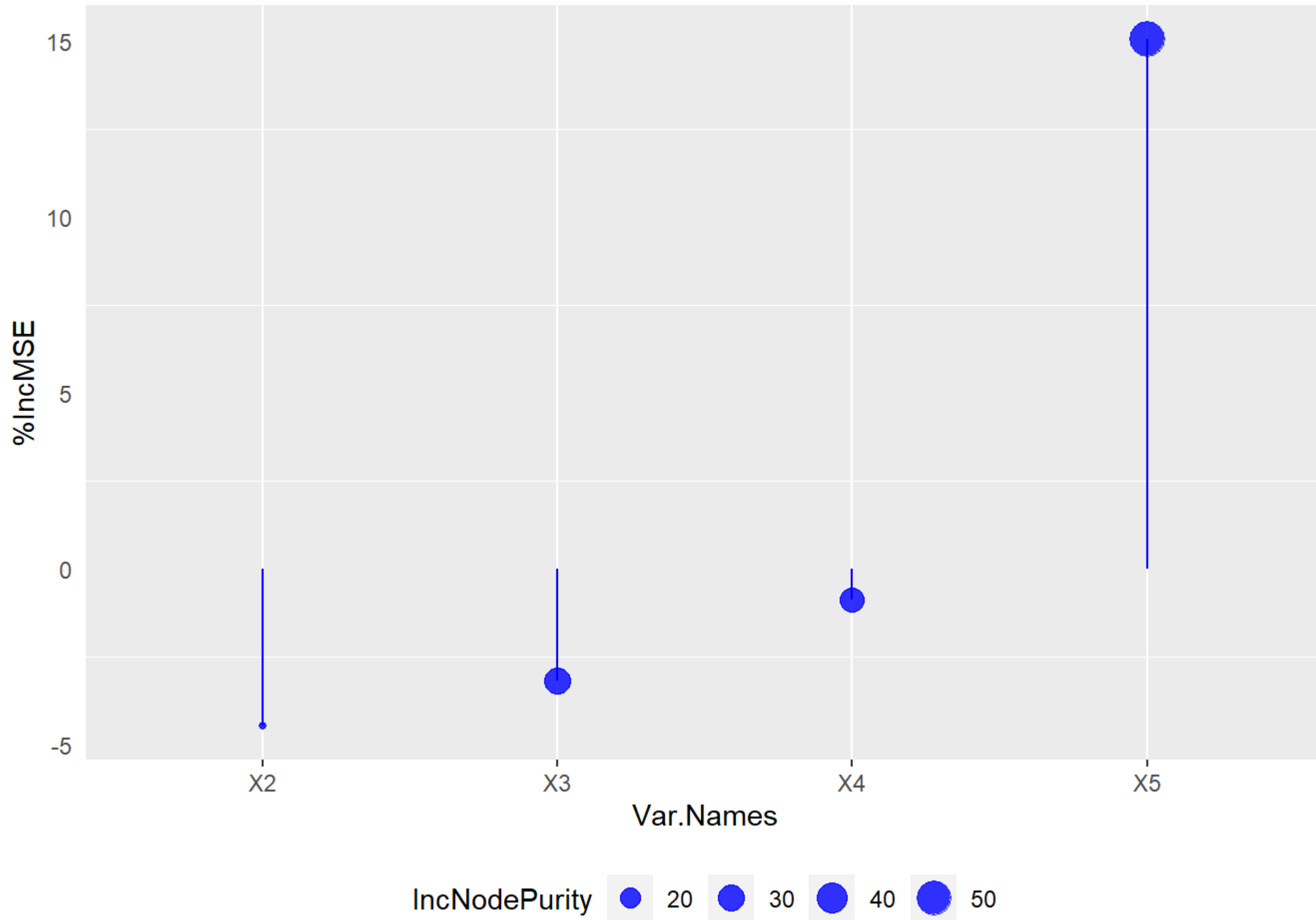


Get

variable importance from the model fit

```
ImpData <- as.data.frame(importance(rf_model))
ImpData$Var.Names <- row.names(ImpData)

ggplot(ImpData, aes(x=Var.Names, y=`%IncMSE`)) +
  geom_segment( aes(x=Var.Names, xend=Var.Names, y=0, yend=`%IncMSE`), color="blue") +
  geom_point(aes(size = IncNodePurity), color="blue", alpha=0.8) +
  #theme_light() +
  #coord_flip() +
  theme(
    legend.position="bottom",
    panel.grid.major.y = element_blank(),
    panel.border = element_blank(),
    axis.ticks.y = element_blank()
  )
```

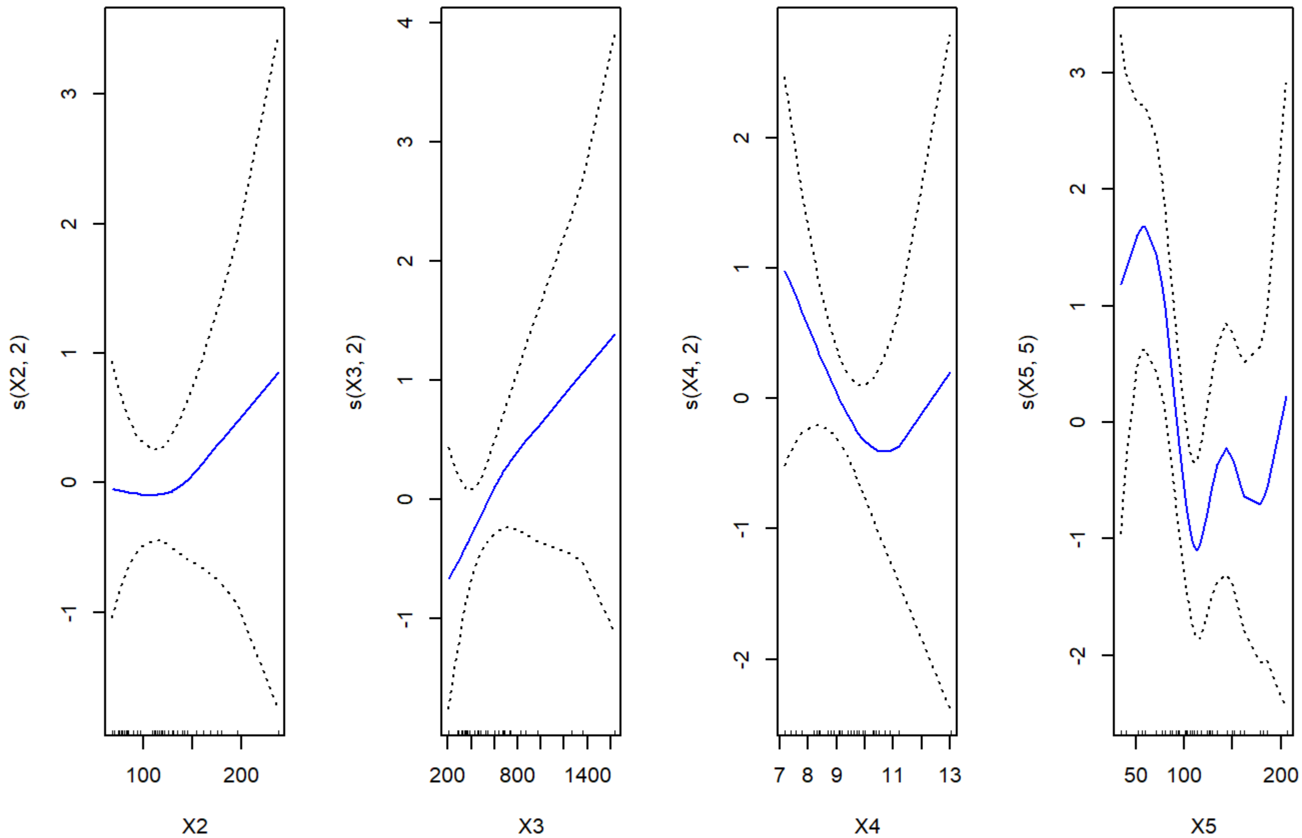


- **Random Forest**
    - Train MSE : 3.401712
    - Test MSE : 0.8120783

GAM

Fitting the GAM Model to our data:

```
gam.m1 = gam(X1~s(X2,4)+s(X3,4)+s(X4,4)+s(X5,4), data = tr_data)
gam.m2 = gam(X1~s(X2,2)+s(X3,2)+s(X4,5)+s(X5,5), data = tr_data)
gam.m3 = gam(X1~s(X2,2)+ s(X3,2)+s(X4,2)+s(X5,5), data = tr_data)
gam.m4 = gam(X1~lo(X2,X3,X4,X5,span=0.5), data = tr_data)
par(mfrow = c(1,4))
plot(gam.m3,se=T,col = 'blue')
```




```
summary(gam.m1)
```

```
##
## Call: gam(formula = X1 ~ s(X2, 4) + s(X3, 4) + s(X4, 4) + s(X5, 4),
##     data = tr_data)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -4.65731 -0.50099  0.09459  0.72751  2.48819
##
## (Dispersion Parameter for gaussian family taken to be 2.9372)
##
##     Null Deviance: 135.7029 on 41 degrees of freedom
## Residual Deviance: 73.4321 on 25.0005 degrees of freedom
## AIC: 178.6549
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##               Df Sum Sq Mean Sq F value  Pr(>F)
## s(X2, 4)   1.000  2.215  2.2155  0.7543 0.39339
## s(X3, 4)   1.000  2.081  2.0810  0.7085 0.40792
## s(X4, 4)   1.000  4.842  4.8417  1.6484 0.21095
## s(X5, 4)   1.000 14.178 14.1776  4.8269 0.03751 *
## Residuals 25.001 73.432  2.9372
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df  Npar F  Pr(F)
## (Intercept)
## s(X2, 4)          3 0.22976 0.8748
## s(X3, 4)          3 0.43897 0.7271
## s(X4, 4)          3 0.64788 0.5916
## s(X5, 4)          3 3.05909 0.0467 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(gam.m2)
```

```
##
## Call: gam(formula = X1 ~ s(X2, 2) + s(X3, 2) + s(X4, 5) + s(X5, 5),
##     data = tr_data)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -4.4103 -0.4183  0.1099  0.6148  2.4989
##
## (Dispersion Parameter for gaussian family taken to be 2.4963)
##
##     Null Deviance: 135.7029 on 41 degrees of freedom
## Residual Deviance: 67.3991 on 26.9997 degrees of freedom
## AIC: 171.0558
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##           Df Sum Sq Mean Sq F value  Pr(>F)
## s(X2, 2)   1  2.246  2.2460  0.8997 0.35127
## s(X3, 2)   1  2.319  2.3185  0.9288 0.34373
## s(X4, 5)   1  5.696  5.6958  2.2817 0.14252
## s(X5, 5)   1 13.468 13.4680  5.3952 0.02797 *
## Residuals 27 67.399  2.4963
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar F   Pr(F)
## (Intercept)
## s(X2, 2)          1 0.3826 0.54140
## s(X3, 2)          1 0.1919 0.66485
## s(X4, 5)          4 1.0207 0.41439
## s(X5, 5)          4 3.5652 0.01849 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(gam.m3)
```

```
##
## Call: gam(formula = X1 ~ s(X2, 2) + s(X3, 2) + s(X4, 2) + s(X5, 5),
##     data = tr_data)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -4.42228 -0.47901  0.08693  0.70358  2.62575
##
## (Dispersion Parameter for gaussian family taken to be 2.4073)
##
##     Null Deviance: 135.7029 on 41 degrees of freedom
## Residual Deviance: 72.2174 on 29.9999 degrees of freedom
## AIC: 167.9554
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##           Df Sum Sq Mean Sq F value  Pr(>F)
## s(X2, 2)   1  2.791  2.7914  1.1596 0.29014
## s(X3, 2)   1  2.146  2.1459  0.8914 0.35264
## s(X4, 2)   1  5.740  5.7395  2.3843 0.13305
## s(X5, 5)   1 13.699 13.6994  5.6909 0.02357 *
## Residuals 30 72.217  2.4073
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar F  Pr(F)
## (Intercept)
## s(X2, 2)          1 0.4577 0.5039
## s(X3, 2)          1 0.2828 0.5988
## s(X4, 2)          1 2.1437 0.1536
## s(X5, 5)          4 3.4719 0.0191 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(gam.m4)
```

```
##
## Call: gam(formula = X1 ~ lo(X2, X3, X4, X5, span = 0.5), data = tr_data)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8179 -0.4614  0.1484  0.6833  2.3563
##
## (Dispersion Parameter for gaussian family taken to be 2.7965)
##
##     Null Deviance: 135.7029 on 41 degrees of freedom
## Residual Deviance: 73.3568 on 26.2314 degrees of freedom
## AIC: 176.15
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                                    Df Sum Sq Mean Sq F value  Pr(>F)
## lo(X2, X3, X4, X5, span = 0.5)  4.000 24.790  6.1974  2.2161 0.09473 .
## Residuals                      26.231 73.357  2.7965
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                                Npar Df Npar F  Pr(F)
## (Intercept)
## lo(X2, X3, X4, X5, span = 0.5)    10.8 1.2471 0.3074
```

AIC crierion for gam.m3 is smaller than that of gam.m2,gam.m1,gam.m4, hence we we will use gam.m3 for model building.

```
anova(gam.m1,gam.m2,gam.m3,test="F")
```

```
## Analysis of Deviance Table
##
## Model 1: X1 ~ s(X2, 4) + s(X3, 4) + s(X4, 4) + s(X5, 4)
## Model 2: X1 ~ s(X2, 2) + s(X3, 2) + s(X4, 5) + s(X5, 5)
## Model 3: X1 ~ s(X2, 2) + s(X3, 2) + s(X4, 2) + s(X5, 5)
##   Resid. Df Resid. Dev      Df Deviance      F Pr(>F)
## 1    25.001     73.432
## 2    27.000     67.399 -1.9992   6.0330
## 3    30.000     72.217 -3.0002  -4.8182 0.5468 0.6549
```

```
pred_tr_gam <- predict(gam.m3,data=tr_data)
tr_mse_gam <- mean((pred_tr_gam - tr_data$X1)^2)
pred_test_gam <- predict(gam.m3,newdata=test_data)
mse_gam <- mean((pred_test_gam - test_data$X1)^2)
ssr_gam <- sum((test_data$X1 - pred_test_gam)^2)
sst_gam <- sum((mean(test_data$X1) - pred_test_gam)^2)
R2_gam <- 1 -(ssr_gam/sst_gam)
```

- **GAM**

- Train MSE : 1.7194609
- Test MSE : 3.2853849
- R sq. : -0.1264827

GLM

LOOCV

```
cv.error <- rep(0, 10)
for (i in 1:10) {
  glm.fit <- glm(X1 ~ X2 + poly(X3,i) + X4+poly(X5 , i), data = data)
  cv.error[i] <- cv.glm(data , glm.fit)$delta [1]
}
cv.error
```

```
##  [1] 2.769386e+00 3.051193e+00 6.486732e+00 1.718157e+01 3.255603e+02
##  [6] 1.982911e+04 1.871400e+04 4.379366e+06 1.195060e+08 7.664137e+09
```

10-fold CV

```
set.seed (99)
cv.error.10 <- rep(0, 10)
for (i in 1:10) {
  glm.fit <- glm(X1 ~ X2 + poly(X3,i) + X4+poly(X5 , i), data = data)
  cv.error.10[i] <- cv.glm(data , glm.fit , K = 10)$delta [1]
}
cv.error.10
```

```
##  [1] 2.881661e+00 3.308143e+00 5.973821e+00 4.526098e+01 6.565030e+02
##  [6] 2.142706e+04 1.377076e+05 3.383585e+07 3.934244e+04 9.097843e+09
```

5_fold CV

```
set.seed (99)
cv.error.5 <- rep(0, 5)
for (i in 1:10) {
  glm.fit <- glm(X1 ~ X2 + poly(X3,i) + X4+poly(X5 , i), data = data)
  cv.error.5[i] <- cv.glm(data , glm.fit , K = 5)$delta [1]
}
cv.error.5
```

```
##  [1] 3.008281e+00 3.661335e+00 4.723311e+00 3.693473e+01 4.171201e+02
##  [6] 8.754775e+04 1.380466e+03 1.524861e+07 1.178687e+10 4.006330e+10
```

SVR

Fitting the SVR Model to our data:

```
descriptors_train_svr = tr_data[,! names(tr_data) %in% c("X1")]
descriptors_test_svr = test_data[,! names(tr_data) %in% c("X1")]
descriptors_train_svr = as.matrix(descriptors_train_svr)
descriptors_test_svr = as.matrix(descriptors_test_svr)
properties_train_svr = tr_data$X1
properties_test_svr = test_data$X1
```

Finding the best model by tuning. Selecting values of epsilon from 0 to 1 at gap of 0.1, Cost from 1 to 10 at gap of 1, gamma from 0.1, 1, 5, 10, 100

```
model_svr = tune(svm, properties_train_svr ~ descriptors_train_svr, ranges=list(epsilon=seq(0,1,
0.1),                    cost=1:10, gamma=c(0.1, 1, 5, 10, 100)))
best_model_svr = model_svr$best.model
summary(best_model_svr)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = properties_train_svr ~ descriptors_train_svr,
##     ranges = list(epsilon = seq(0, 1, 0.1), cost = 1:10, gamma = c(0.1,
##         1, 5, 10, 100)))
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  2
##       gamma:  0.1
##     epsilon:  0.2
##
##
## Number of Support Vectors:  32
```

Using the tuning parameters from the best model

```
kernel = 'radial'
cost = best_model_svr$cost
gamma = best_model_svr$gamma
epsilon = best_model_svr$epsilon

svm_fit = svm(tr_data$X1 ~ ., data = tr_data, method = 'eps-regression', kernel = kernel, cost =
cost,                gamma = gamma, epsilon = epsilon)
pred_train_svr = predict(svm_fit, data = descriptors_train_svr)
pred_test_svr = predict(svm_fit, newdata = descriptors_test_svr)
tr_mse_svr = mean((pred_train_svr - tr_data$X1)^2)
mse_svr = mean((pred_test_svr - test_data$X1)^2)
ssr_svr = sum((pred_train_svr - tr_data$X1)^2)
R2_svr <- 1 -(ssr_svr/sst)
```

This model resulted in Train MSE : 2.0976561 Test MSE : 1.1008407 R sq. : 0.3662247 As this is not a good fit so tuning the model further manually to find the best model by changing cost, gamma and epsilon. This resulted in cost = 5 gamma = 5.5 and epsilon = 0.4 for radial kernel.

```
kernel = 'radial'
cost = 5
gamma = 5.5
epsilon = 0.4

svm_fit = svm(tr_data$X1 ~ ., data = tr_data, method = 'eps-regression', kernel = kernel, cost =
cost,                gamma = gamma, epsilon = epsilon)
pred_train_svr = predict(svm_fit, data = descriptors_train_svr)
pred_test_svr = predict(svm_fit, newdata = descriptors_test_svr)
tr_mse_svr = mean((pred_train_svr - tr_data$X1)^2)
mse_svr = mean((pred_test_svr - test_data$X1)^2)
ssr_svr = sum((pred_train_svr - tr_data$X1)^2)
R2_svr <- 1 -(ssr_svr/sst)
```

- **SVR**
  - Train MSE : 0.4304361
  - Test MSE : 0.431499
  - R sq. : 0.8699502

```
results_model
```

```
##          Model_Name        Train_MSE Test_MSE
## 1      Linear Model 2.64079150899683 1.589009
## 2 Polynomial Model 2.24060093752846 1.466093
## 3     Random Forest 2.24060093752846 1.466093
## 4               GAM  1.7194608785101 3.285385
## 5        GLM(LOOCV)             2.60 2.769386
## 6               SVR 0.43043609837064 0.431499
```

Airfoil Self-Noise Data Set

Loading data from the provided file in data.

```
Pilot_data_set = read.table("~/airfoil_self_noise.dat", sep="\t")
```

#viewing the data set

```
View(Pilot_data_set)
```

#Summary of the data set

```
head(Pilot_data_set)
```

```
##     V1 V2     V3   V4         V5      V6
## 1  800  0 0.3048 71.3 0.00266337 126.201
## 2 1000  0 0.3048 71.3 0.00266337 125.201
## 3 1250  0 0.3048 71.3 0.00266337 125.951
## 4 1600  0 0.3048 71.3 0.00266337 127.591
## 5 2000  0 0.3048 71.3 0.00266337 127.461
## 6 2500  0 0.3048 71.3 0.00266337 125.571
```

```
summary(Pilot_data_set)
```

```
##        V1              V2               V3               V4
##  Min.   :  200   Min.   : 0.000   Min.   :0.0254   Min.   :31.70
##  1st Qu.:  800   1st Qu.: 2.000   1st Qu.:0.0508   1st Qu.:39.60
##  Median : 1600   Median : 5.400   Median :0.1016   Median :39.60
##  Mean   : 2886   Mean   : 6.782   Mean   :0.1365   Mean   :50.86
##  3rd Qu.: 4000   3rd Qu.: 9.900   3rd Qu.:0.2286   3rd Qu.:71.30
##  Max.   :20000   Max.   :22.200   Max.   :0.3048   Max.   :71.30
##        V5                  V6
##  Min.   :0.0004007   Min.   :103.4
##  1st Qu.:0.0025351   1st Qu.:120.2
##  Median :0.0049574   Median :125.7
##  Mean   :0.0111399   Mean   :124.8
##  3rd Qu.:0.0155759   3rd Qu.:130.0
##  Max.   :0.0584113   Max.   :141.0
```

where: V1 = Frequency, in Hertz. V2 = The angle of attack, in degrees. V3 = Chord length, in meters. V4 = Free-stream velocity, in meters per second. V5 = Suction side displacement thickness, in meters. V6 = Scaled sound pressure level, in decibels.

#Checking for missing data in every column

```
any(is.na(Pilot_data_set))
```

```
## [1] FALSE
```

There is no missing data in the provided file.

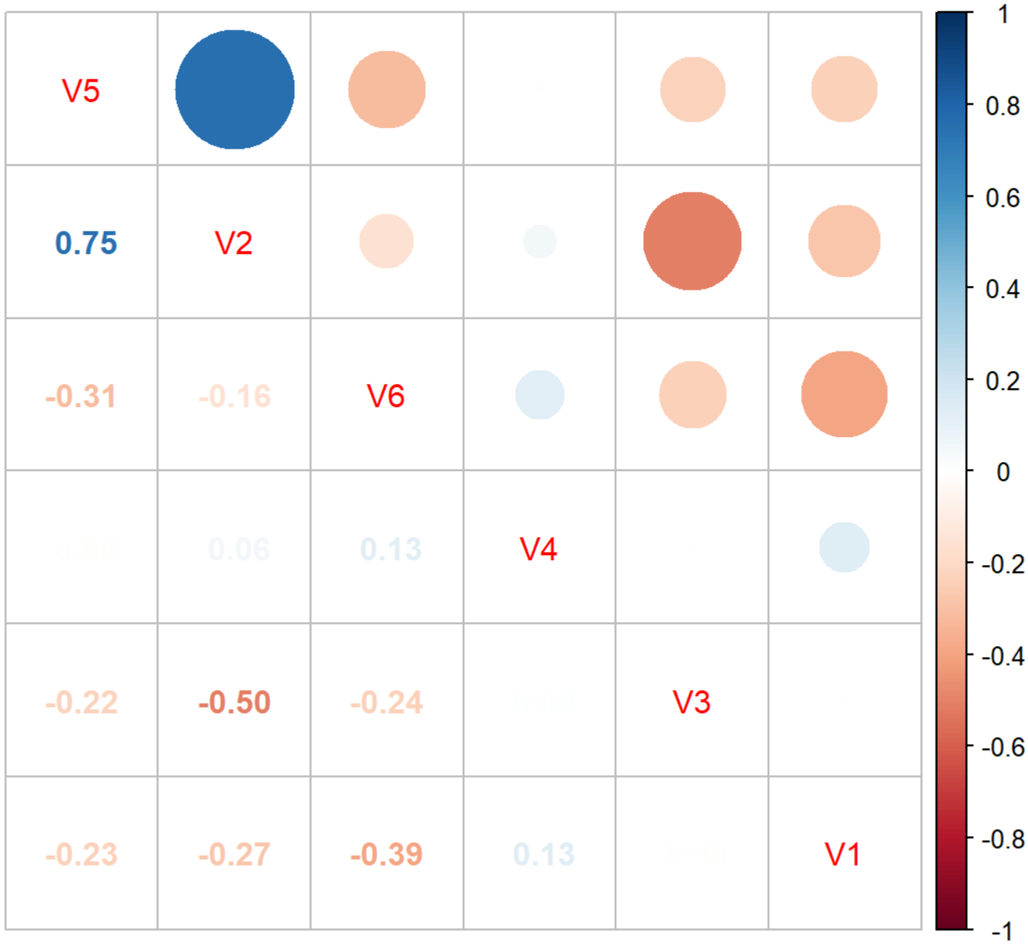
Plotting the data: respose vs every predictors and analysing using plot

Plotting correlation plot and a column vs every other column

```
##              V1          V2           V3           V4           V5         V6
## V1  1.000000000 -0.27276454 -0.003660639  0.133663831 -0.230107353 -0.3907114
## V2 -0.272764536  1.00000000 -0.504868150  0.058759565  0.753393785 -0.1561075
## V3 -0.003660639 -0.50486815  1.000000000  0.003786629 -0.220842431 -0.2361615
## V4  0.133663831  0.05875957  0.003786629  1.000000000 -0.003974013  0.1251028
## V5 -0.230107353  0.75339378 -0.220842431 -0.003974013  1.000000000 -0.3126695
## V6 -0.390711412 -0.15610753 -0.236161512  0.125102801 -0.312669506  1.0000000
```

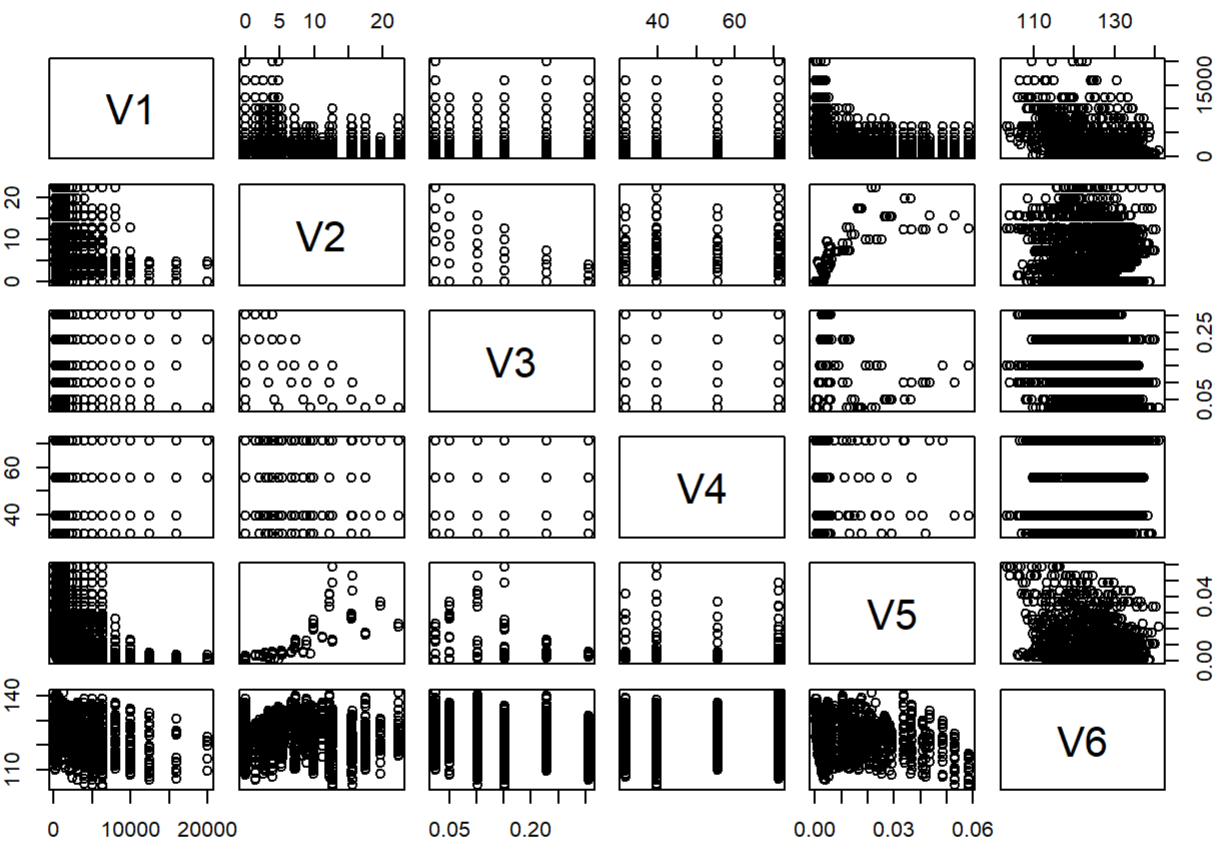
This shows that V1,V2, V3, V5 has a strong negative correlation to V6 and V4 has positive correlation with V6. where V6 is our Response.

```
library(corrplot)
cor_data = cor(Pilot_data_set)
corrplot.mixed(cor(Pilot_data_set), order = 'AOE')
```



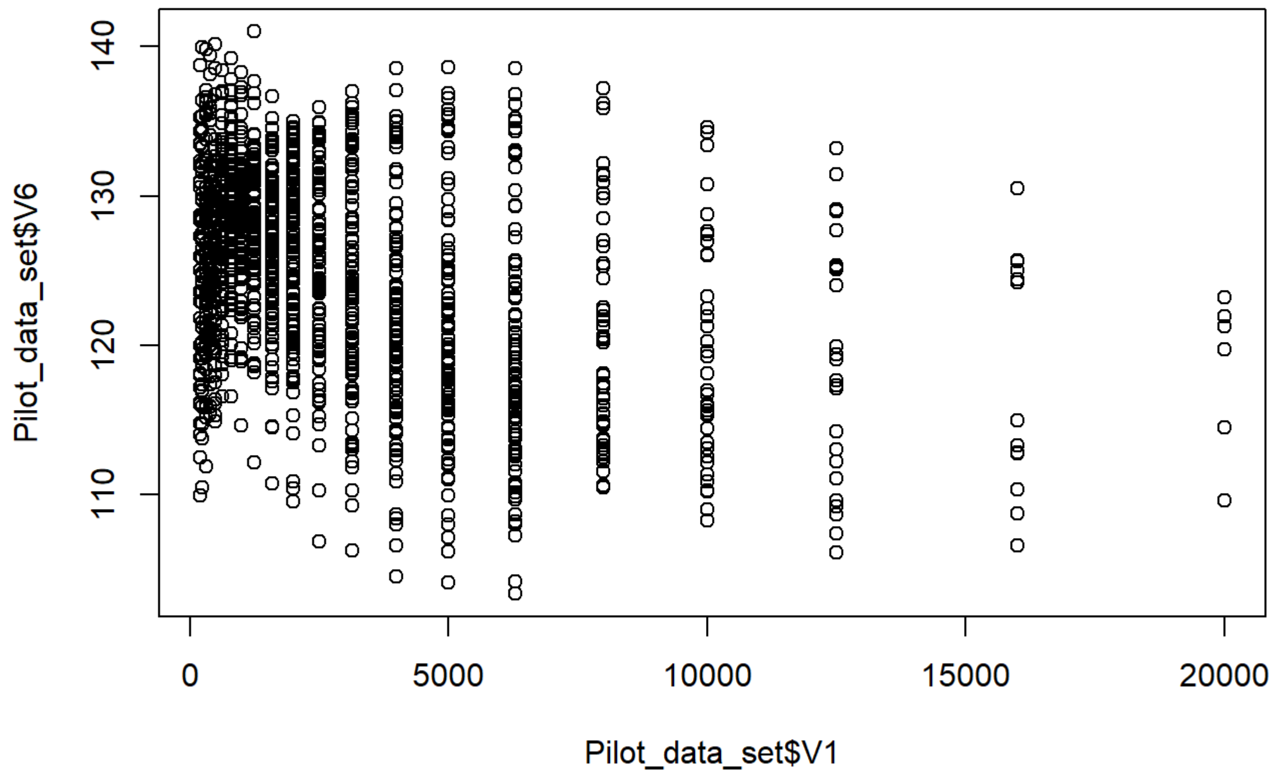
```
#corrplot(cor_data, method = 'color')
```

```
pairs(Pilot_data_set)
```

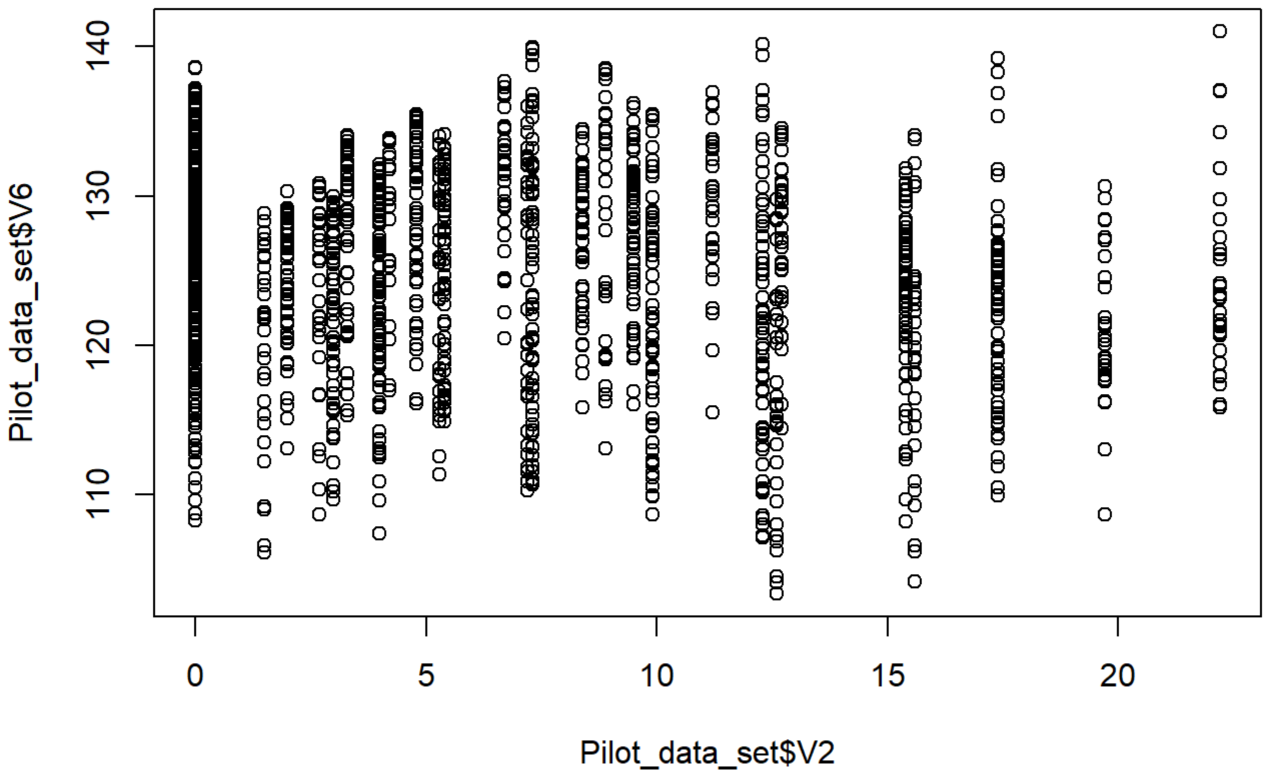


Without scaling

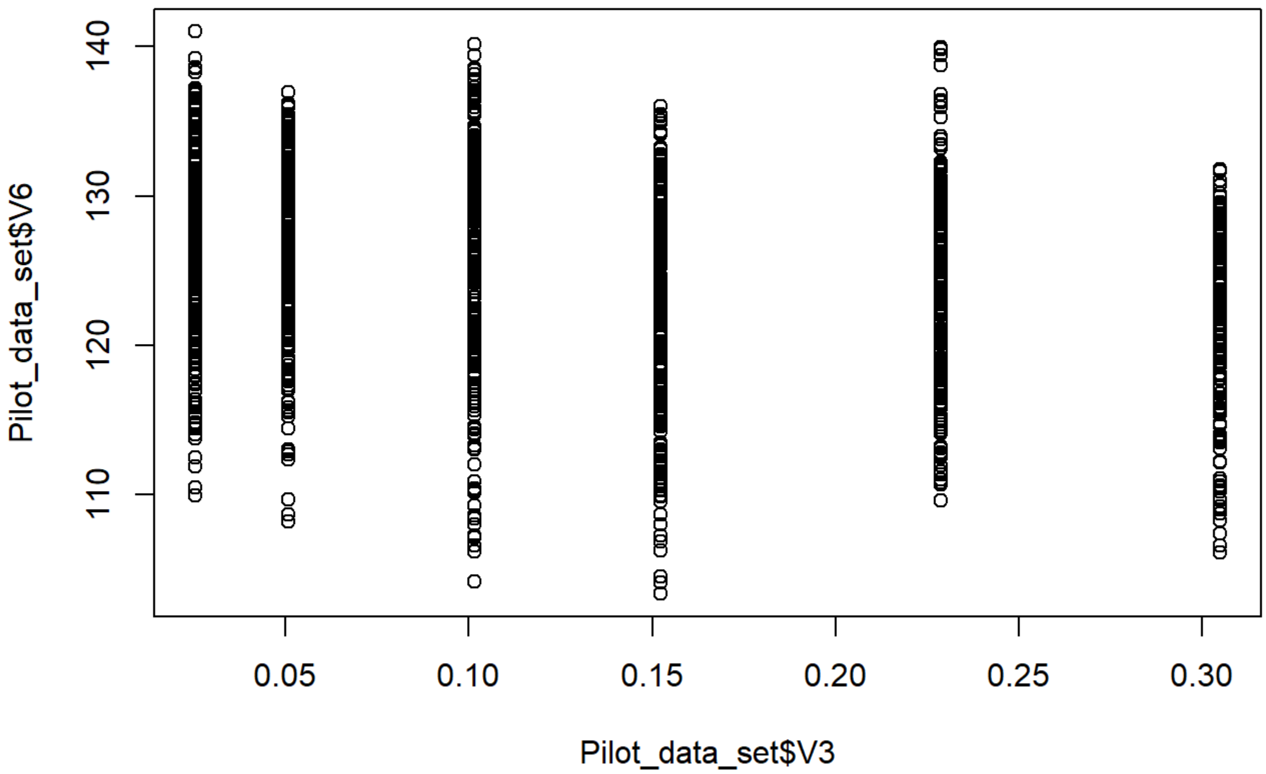
```
plot(Pilot_data_set$V1, Pilot_data_set$V6)
```



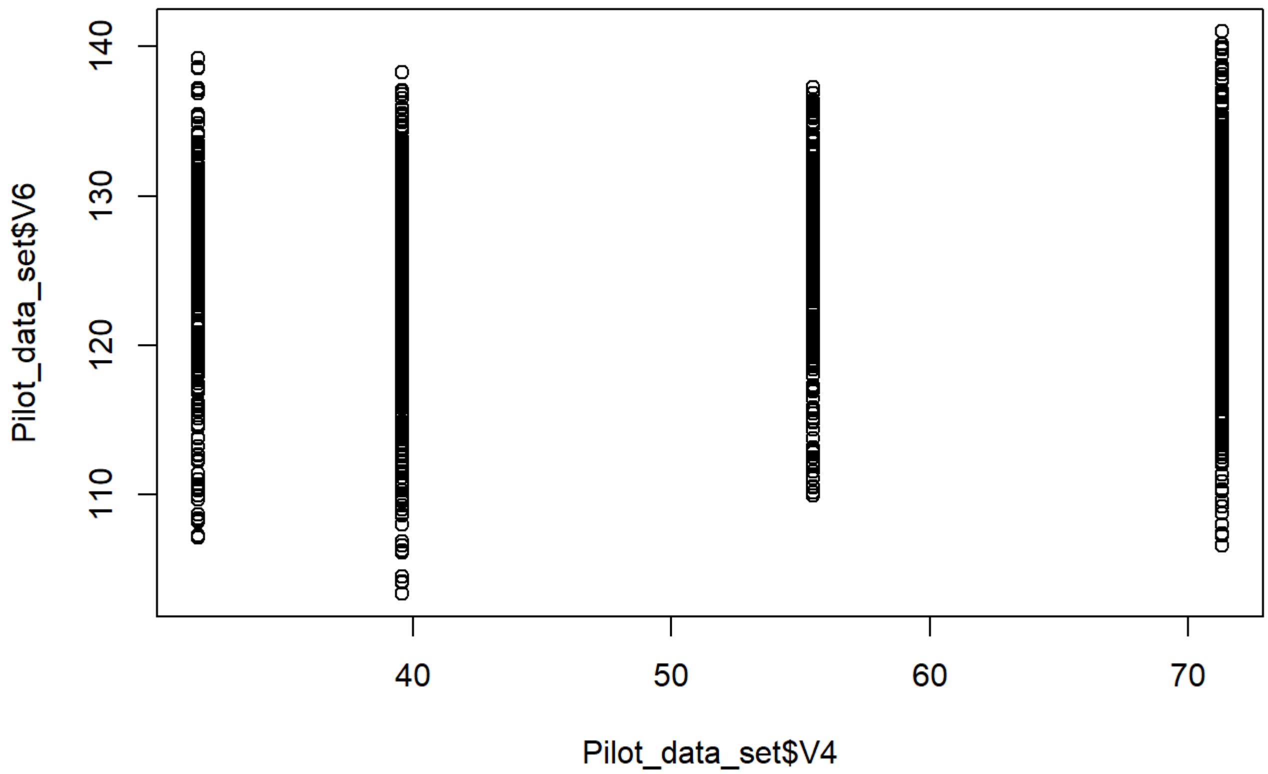

```
plot(Pilot_data_set$V2, Pilot_data_set$V6)
```



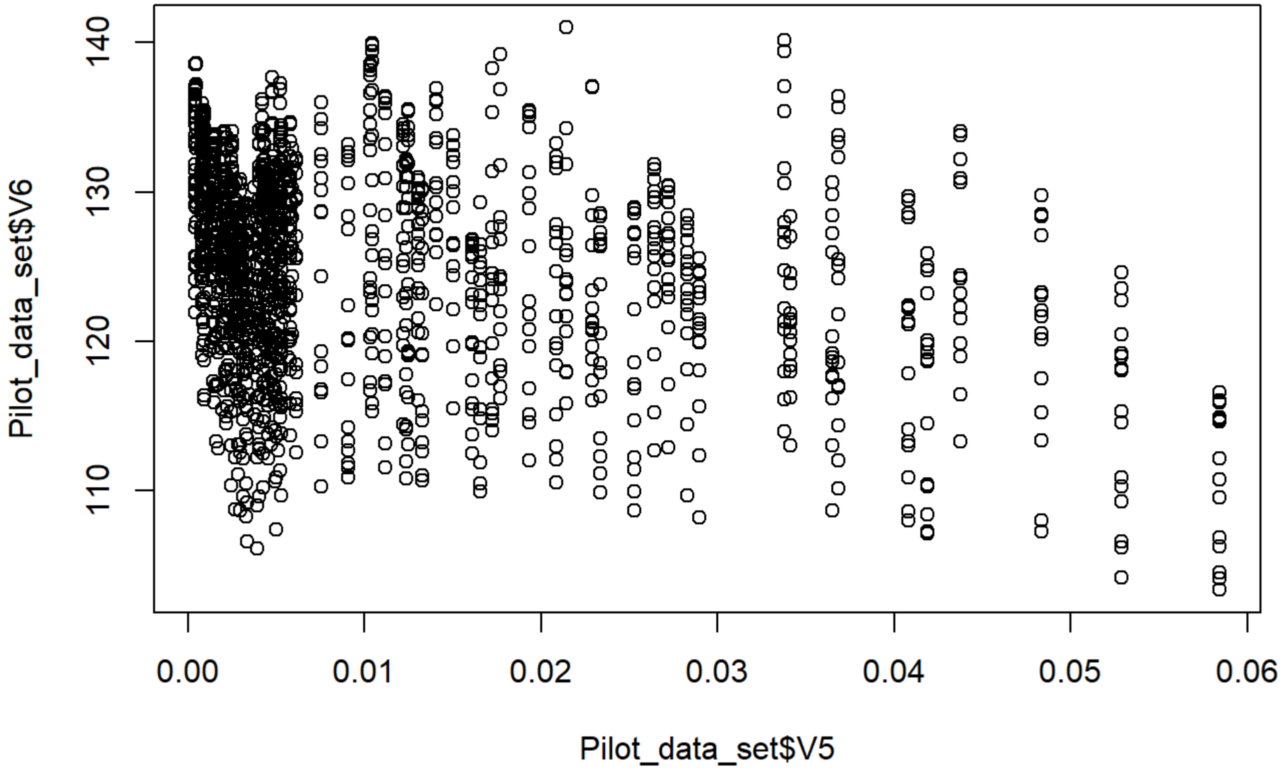

```
plot(Pilot_data_set$V3, Pilot_data_set$V6)
```




```
plot(Pilot_data_set$V4, Pilot_data_set$V6)
```




```
plot(Pilot_data_set$V5, Pilot_data_set$V6)
```

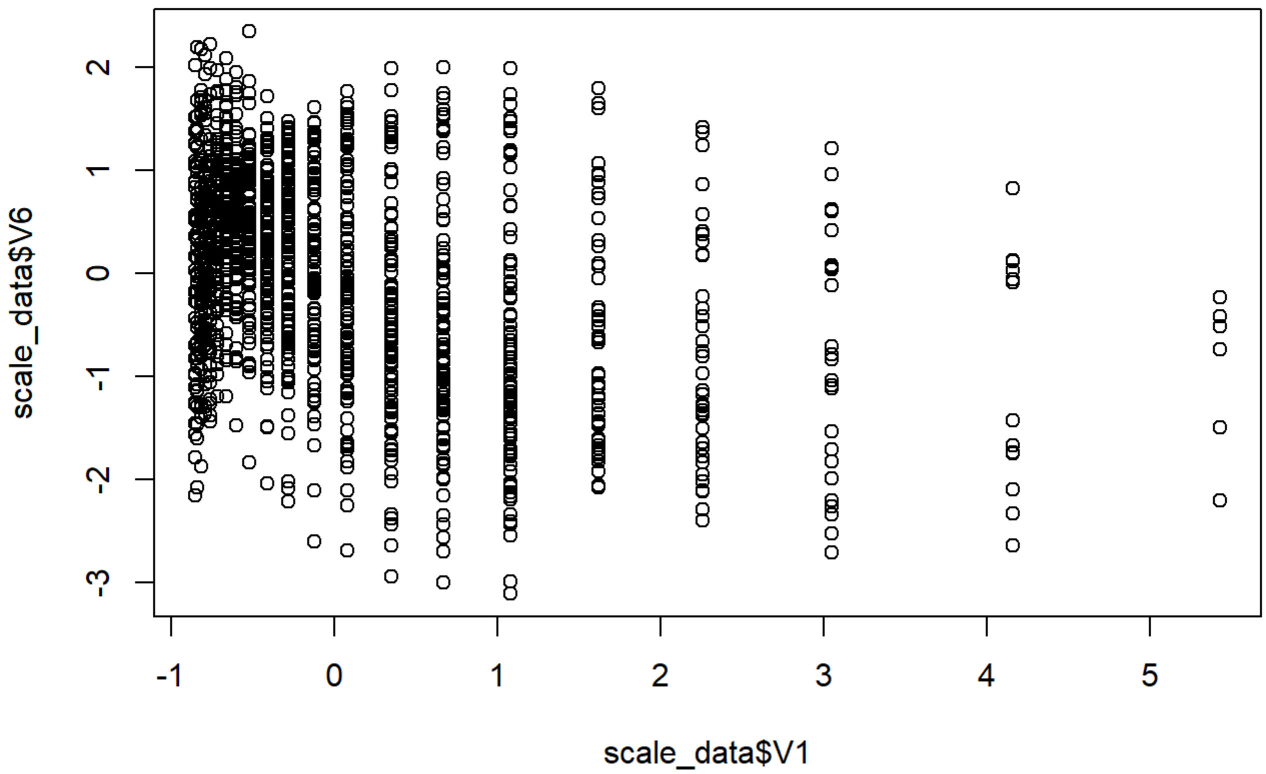


After scaling

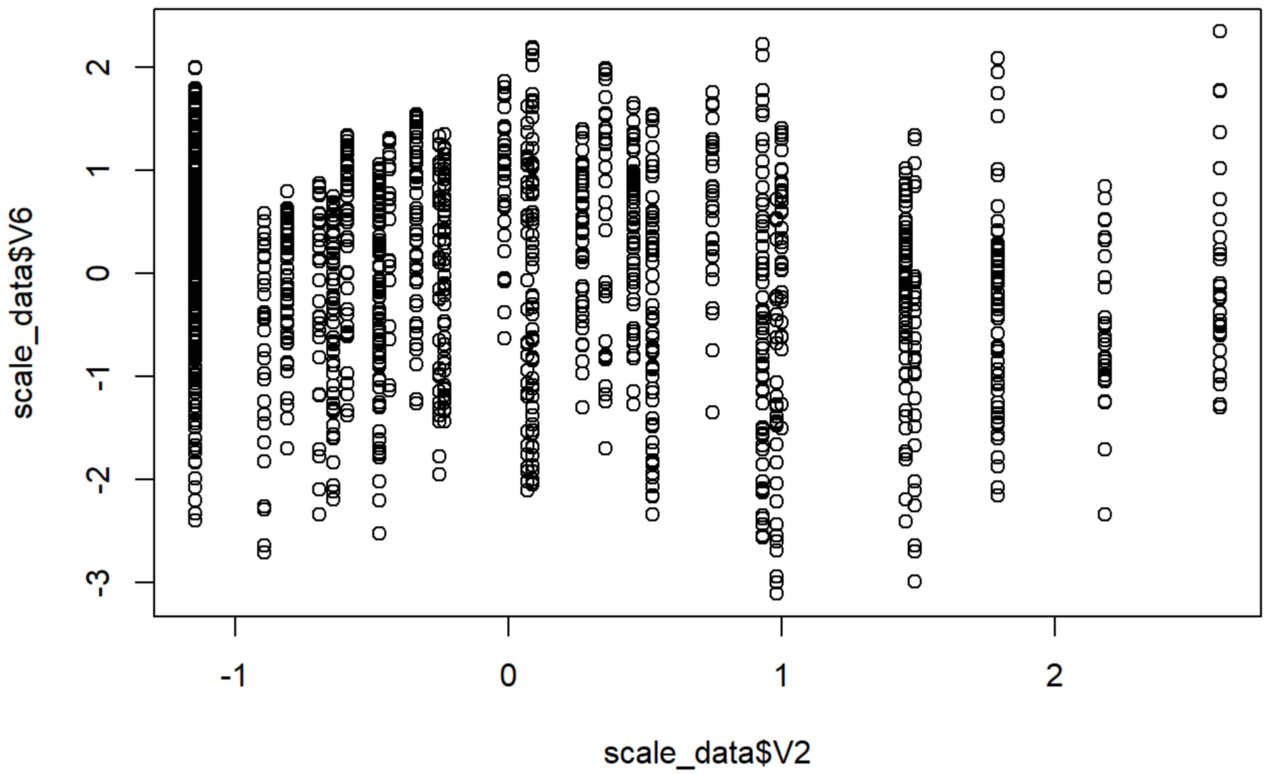
```
scale_data <-as.data.frame(scale(Pilot_data_set))
summary(scale_data)
```

```
##        V1                V2                V3                V4         
##  Min.   :-0.8521   Min.   :-1.1460   Min.   :-1.1882   Min.   :-1.2304  
##  1st Qu.:-0.6618   1st Qu.:-0.8081   1st Qu.:-0.9167   1st Qu.:-0.7231  
##  Median :-0.4080   Median :-0.2336   Median :-0.3736   Median :-0.7231  
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000  
##  3rd Qu.: 0.3532   3rd Qu.: 0.5268   3rd Qu.: 0.9841   3rd Qu.: 1.3125  
##  Max.   : 5.4285   Max.   : 2.6052   Max.   : 1.7987   Max.   : 1.3125  
##        V5                V6         
##  Min.   :-0.8167   Min.   :-3.1102  
##  1st Qu.:-0.6543   1st Qu.:-0.6733  
##  Median :-0.4701   Median : 0.1283  
##  Mean   : 0.0000   Mean   : 0.0000  
##  3rd Qu.: 0.3373   3rd Qu.: 0.7479  
##  Max.   : 3.5947   Max.   : 2.3412  
```

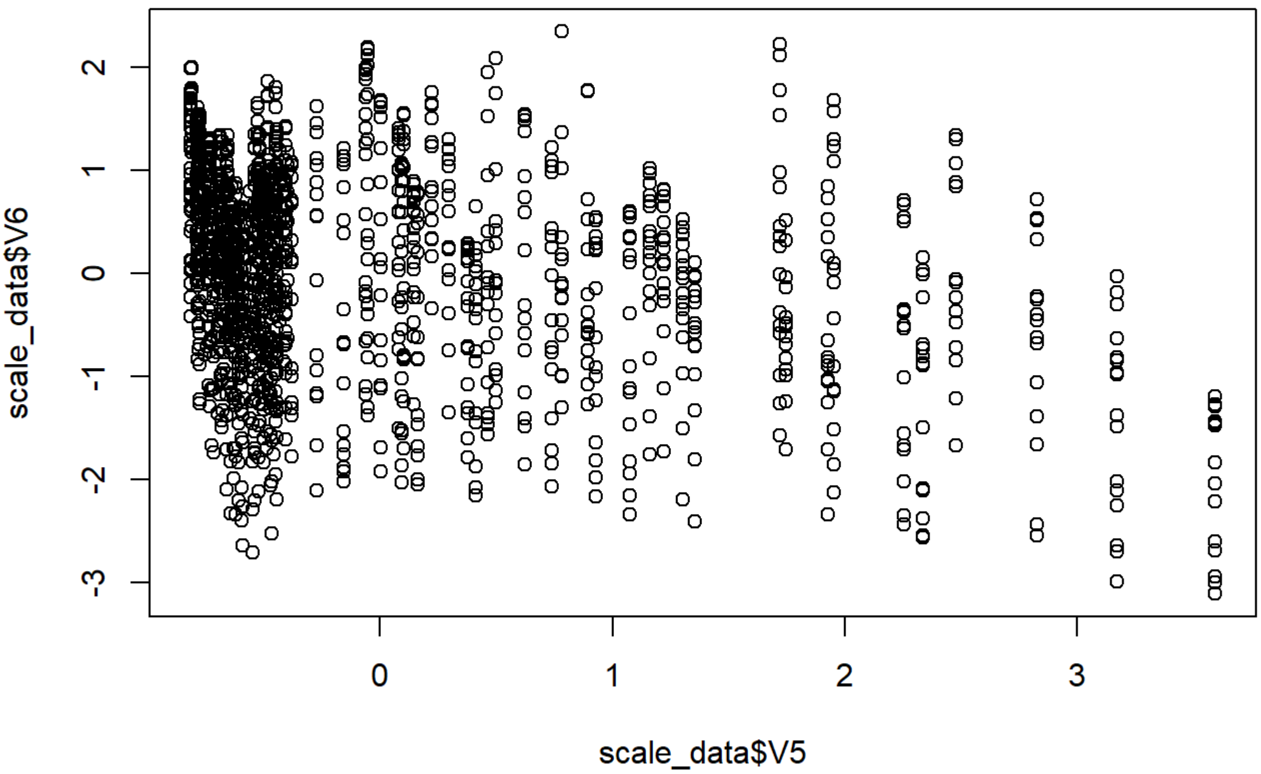
```
plot(scale_data$V1, scale_data$V6)
```



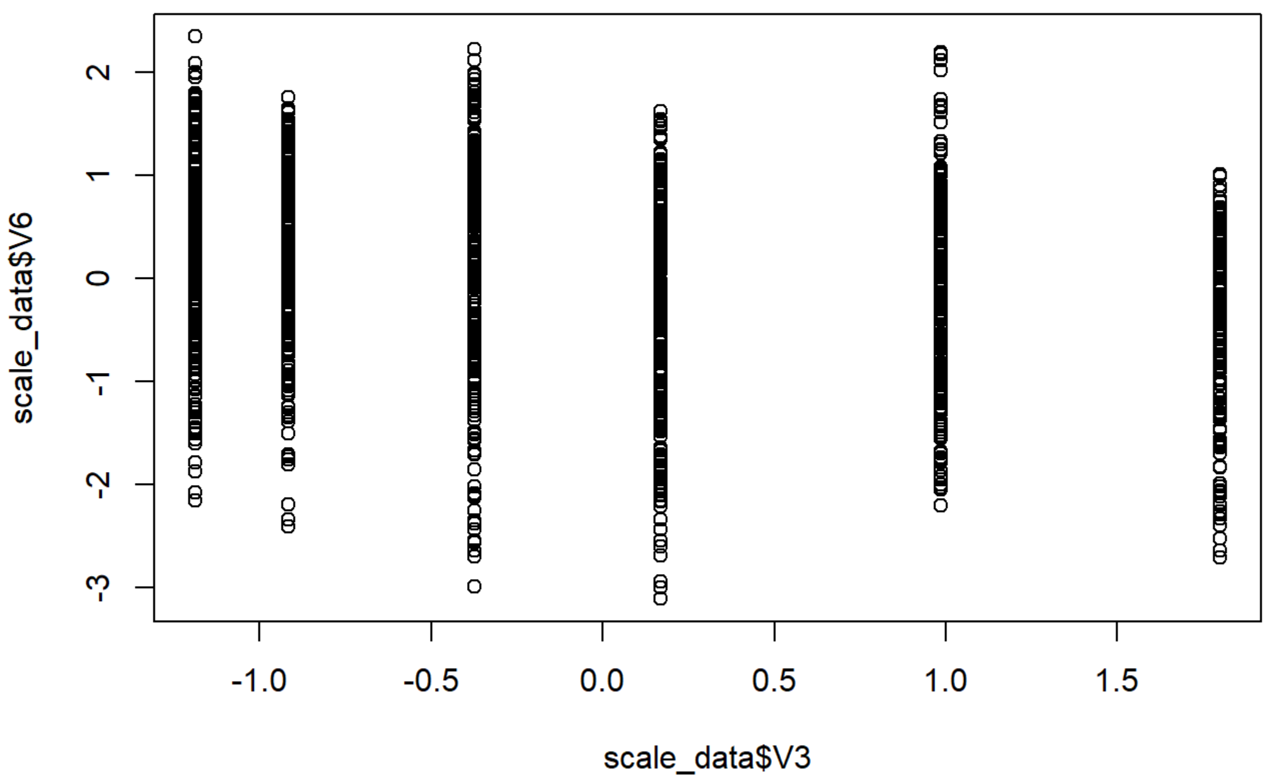

```
plot(scale_data$V2, scale_data$V6)
```



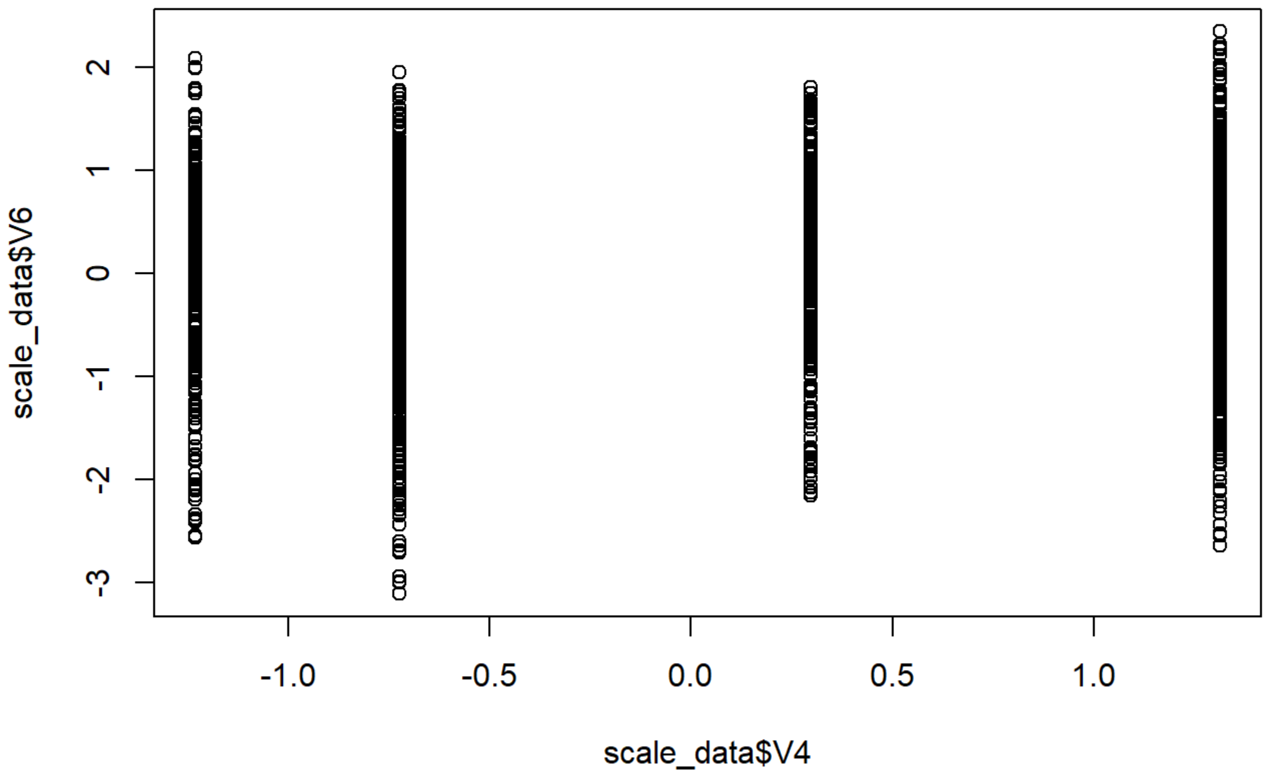

```
plot(scale_data$V5, scale_data$V6)
```



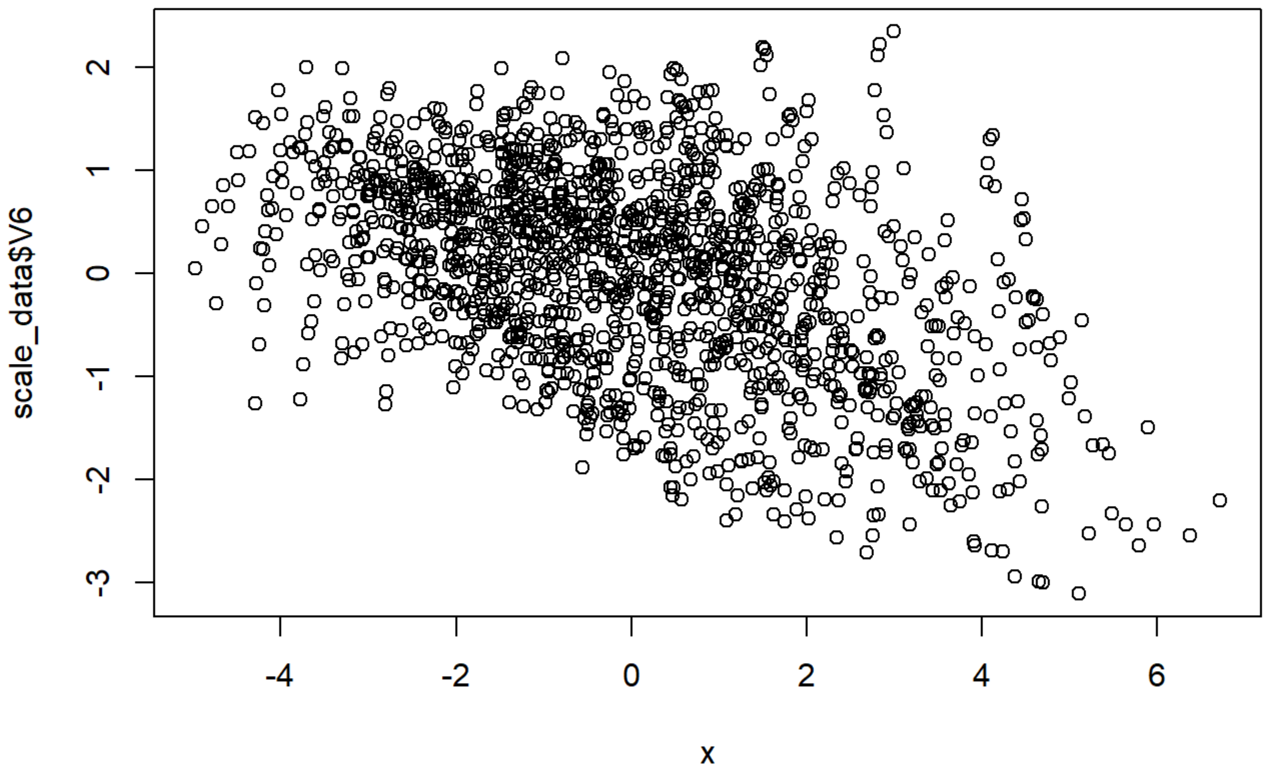

```
plot(scale_data$V3, scale_data$V6)
```




```
plot(scale_data$V4, scale_data$V6)
```




```
x = scale_data$V1+scale_data$V2+scale_data$V3+scale_data$V4+scale_data$V5
plot(x, scale_data$V6)
```



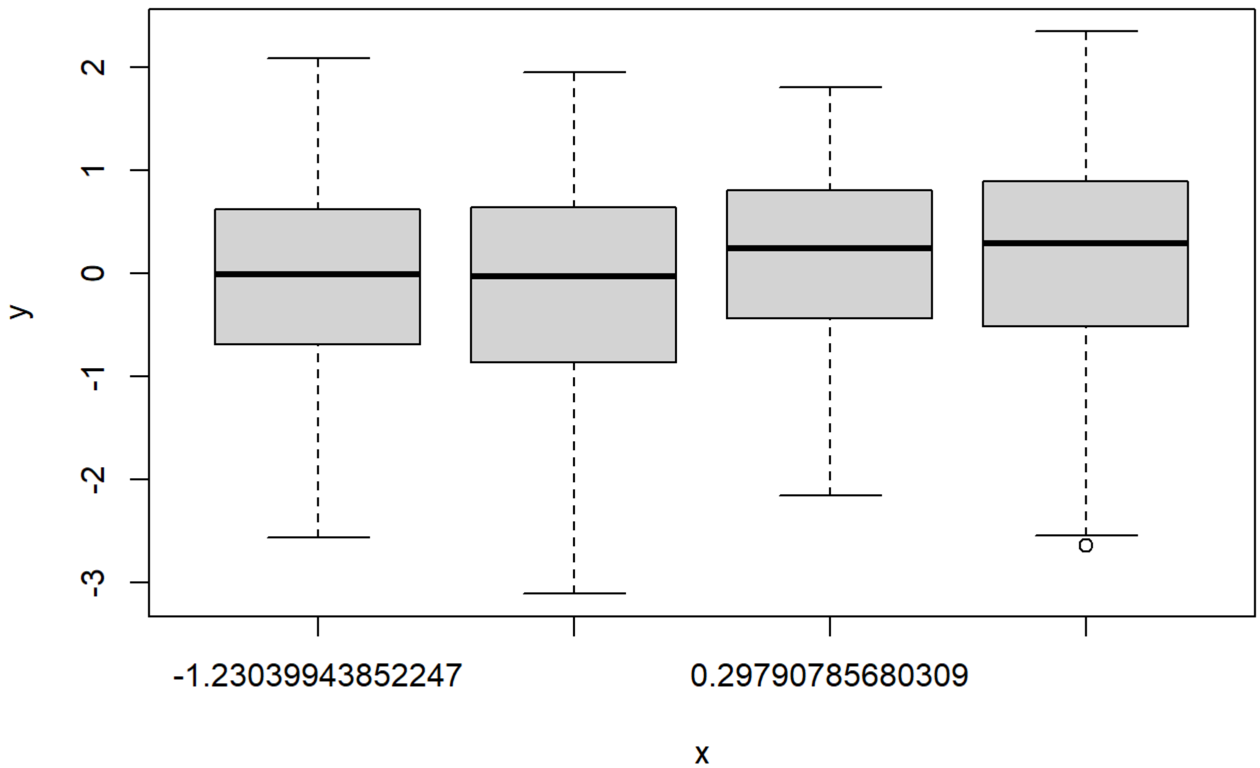

```
#abline(Mul_linear_fit, lwd = 3, col = "red")
```

The V3, V4 variable is stored as a numeric vector, so R has treated it as quantitative. However, since there are only a small number of possible values for V3, V4, one may prefer to treat it as a qualitative variable.

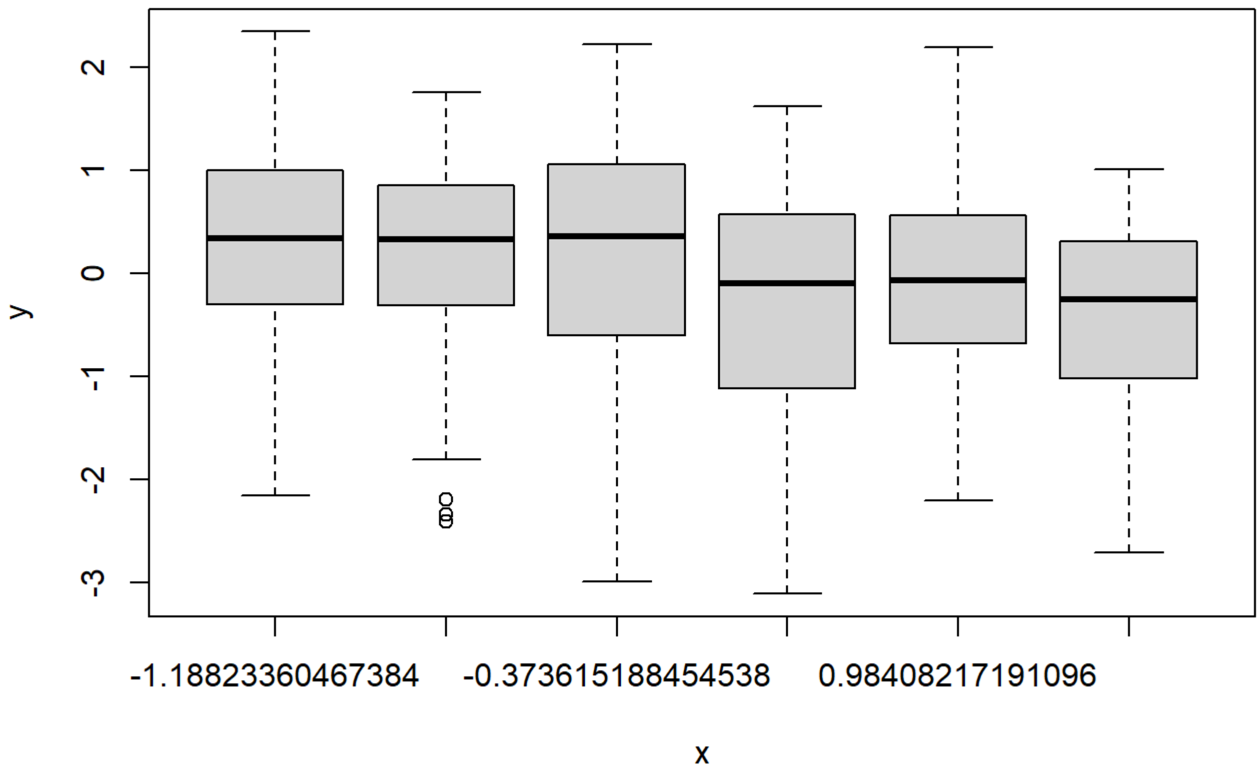
#converting categorical/qualitative variables to quantitative

```
scale_data$V4 <- as.factor(scale_data$V4)
scale_data$V3 <- as.factor(scale_data$V3)
```

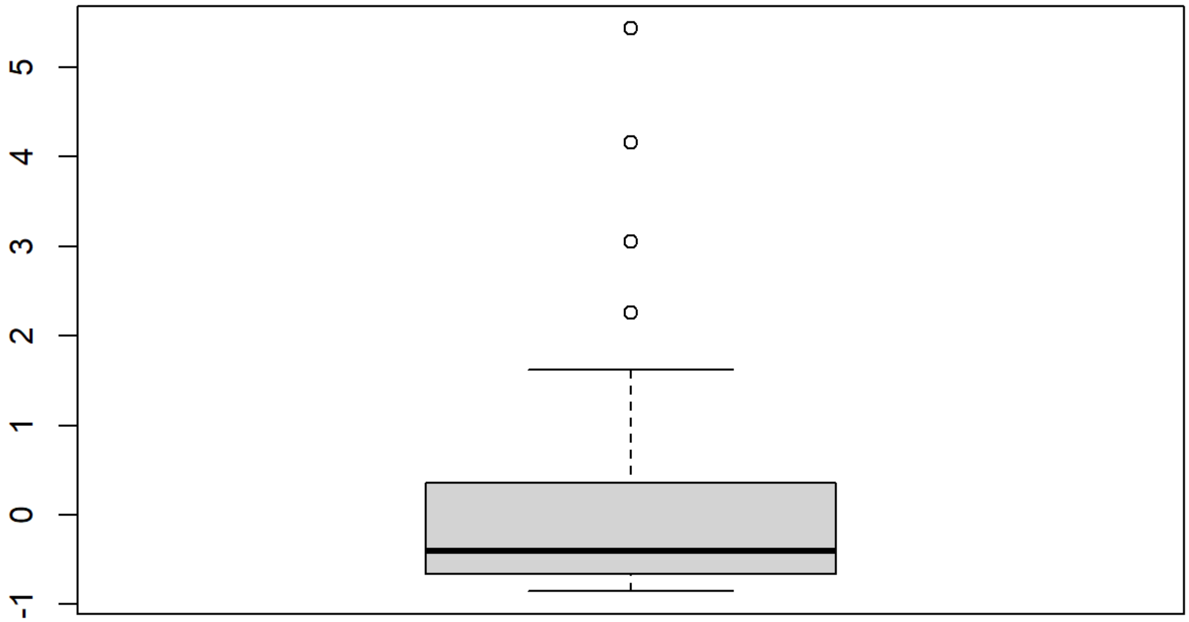
```
plot(scale_data$V4, scale_data$V6)
```



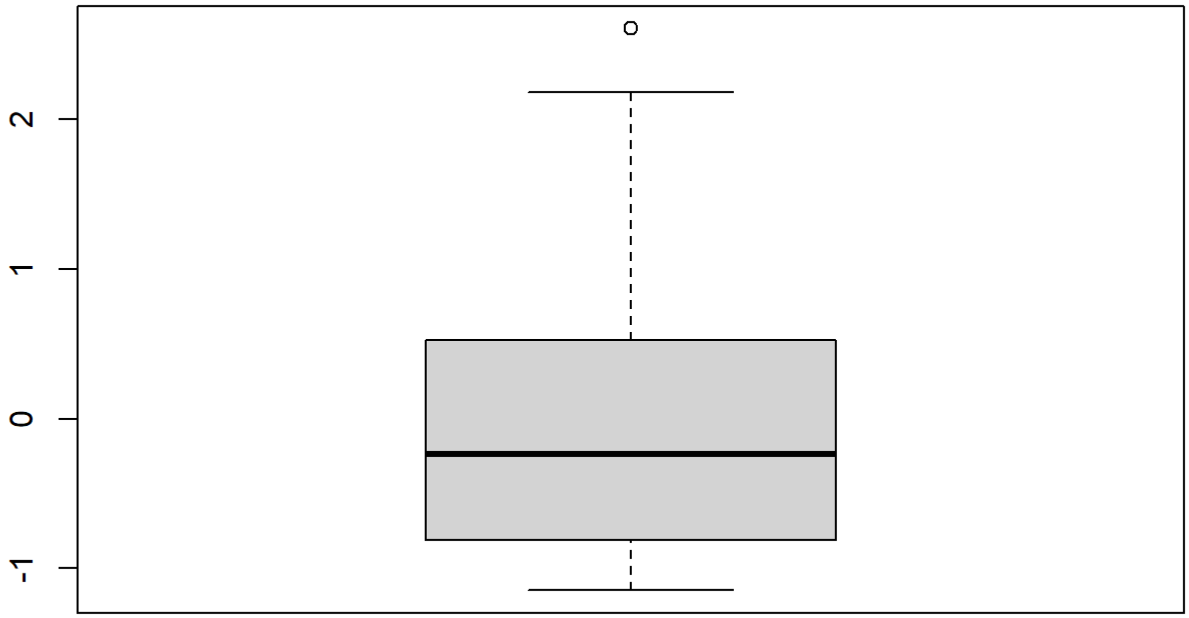

```
plot(scale_data$V3, scale_data$V6)
```



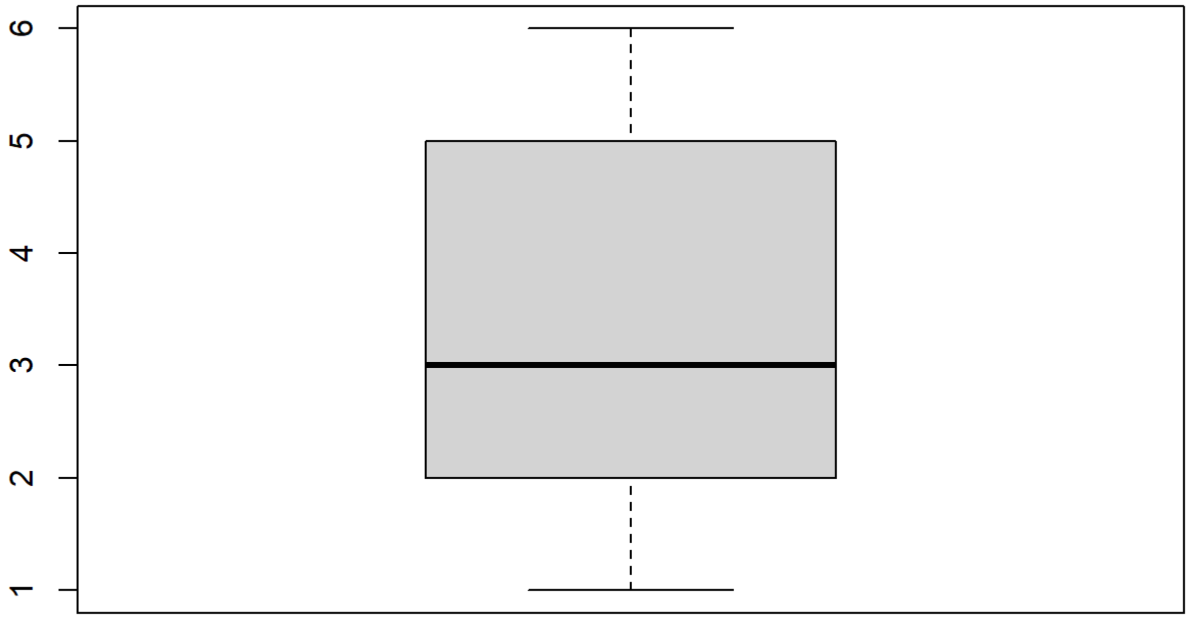

```
boxplot(scale_data$V1)
```



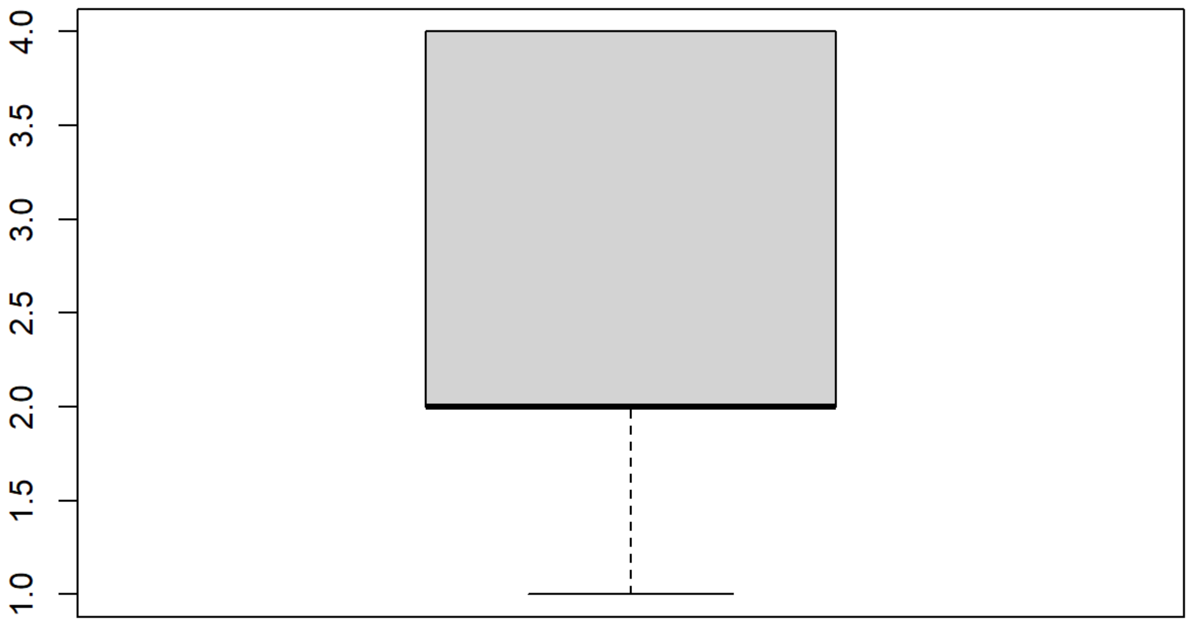

```
boxplot(scale_data$V2)
```



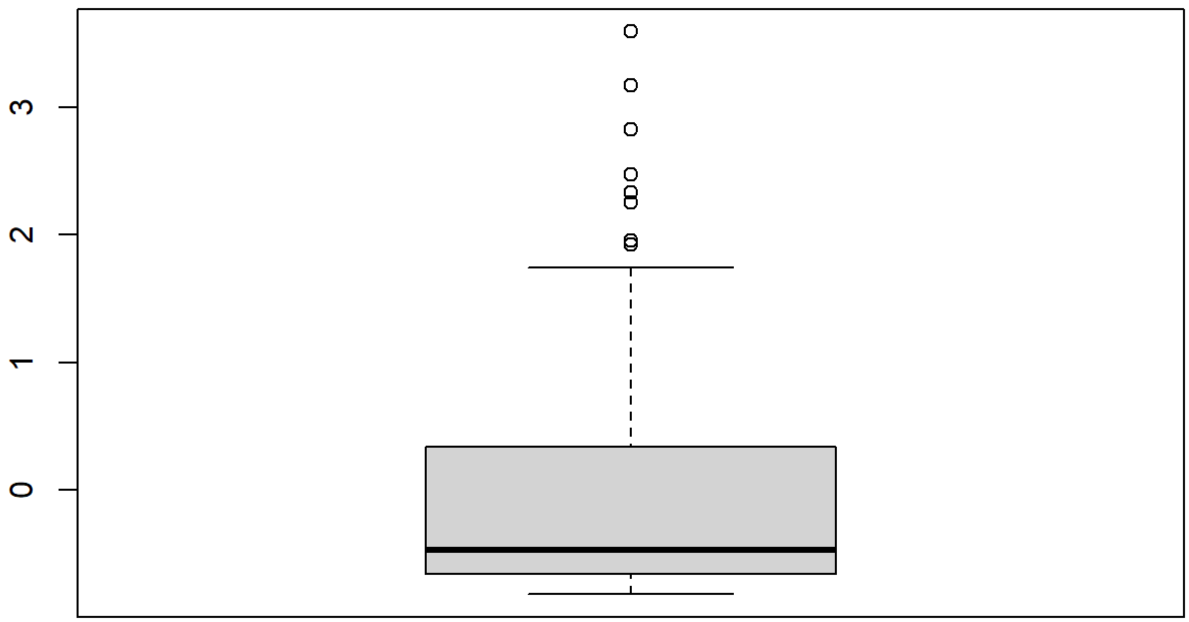

```
boxplot(scale_data$V3)
```




```
boxplot(scale_data$V4)
```




```
boxplot(scale_data$V5)
```



From Boxplot we can clearly see that there are outliers.

###Splitting the data set into Training and Test sets

```
#make this example reproducible
set.seed(1)
#use 70% of dataset as training set and 30% as test set
sample <- sample.split(scale_data$V6, SplitRatio = 0.7)
train  <- subset(scale_data, sample == TRUE)
test   <- subset(scale_data, sample == FALSE)
```

```
prop.table(table(train$V3))
```

```
## 
##  -1.18823360467384 -0.916694132600736 -0.373615188454538  0.169463755691661 
##          0.1901141          0.1501901          0.1768061          0.1777567 
##   0.98408217191096   1.79870058813026 
##          0.1806084          0.1245247
```

```
prop.table(table(test$V3))
```

```
##
##  -1.18823360467384 -0.916694132600736 -0.373615188454538  0.169463755691661
##          0.1729490          0.1751663          0.1707317          0.1862528
##   0.98408217191096   1.79870058813026
##          0.1685144          0.1263858
```

we can see equal split of data sets….

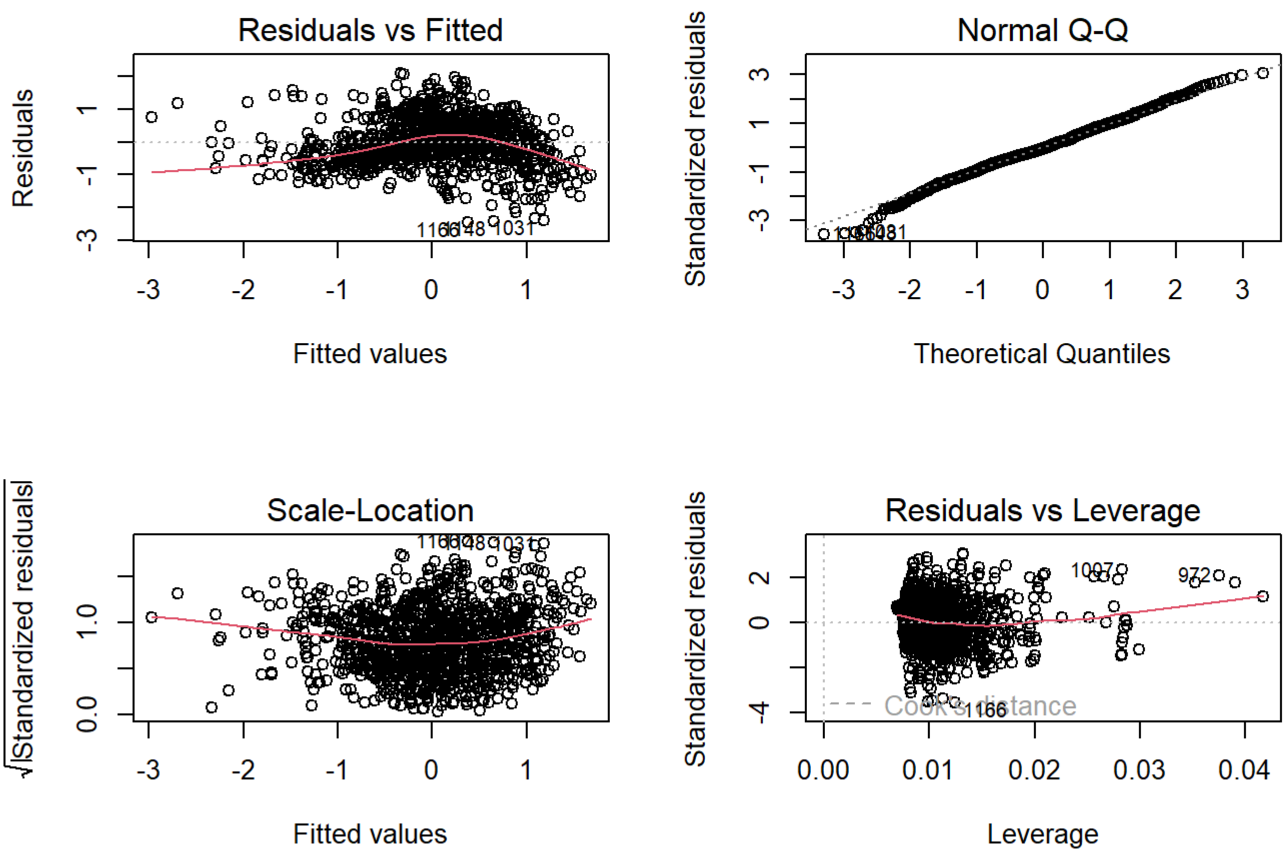
#Multi-linear Regression

```
mul_linear_fit = lm(V6~., data = train)
summary(mul_linear_fit)
```

```
##
## Call:
## lm(formula = V6 ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.46302 -0.41905 -0.02272  0.47110  2.10583
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           0.33302    0.07252   4.592 4.92e-06 ***
## V1                   -0.58115    0.02390 -24.317  < 2e-16 ***
## V2                   -0.35440    0.04552  -7.785 1.67e-14 ***
## V3-0.916694132600736 -0.15120    0.07901  -1.914 0.055951 .
## V3-0.373615188454538 -0.32591    0.08801  -3.703 0.000224 ***
## V30.169463755691661  -0.85535    0.08702  -9.829  < 2e-16 ***
## V30.98408217191096   -1.00428    0.08528 -11.777  < 2e-16 ***
## V31.79870058813026   -1.37282    0.09645 -14.233  < 2e-16 ***
## V4-0.723104159821969  0.09602    0.06284   1.528 0.126822
## V40.29790785680309    0.30292    0.07097   4.268 2.15e-05 ***
## V41.31249841420409    0.55113    0.06557   8.405  < 2e-16 ***
## V5                   -0.24812    0.04144  -5.987 2.94e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.697 on 1040 degrees of freedom
## Multiple R-squared:  0.5053, Adjusted R-squared:  0.5001
## F-statistic: 96.57 on 11 and 1040 DF,  p-value: < 2.2e-16
```

Mul_linear_fit has a RSE of 0.7011; R2 = 0.4965(49%)

```
par(mfrow = c(2,2))
plot(mul_linear_fit)
```



residual plot shows that there are leverage points as wel

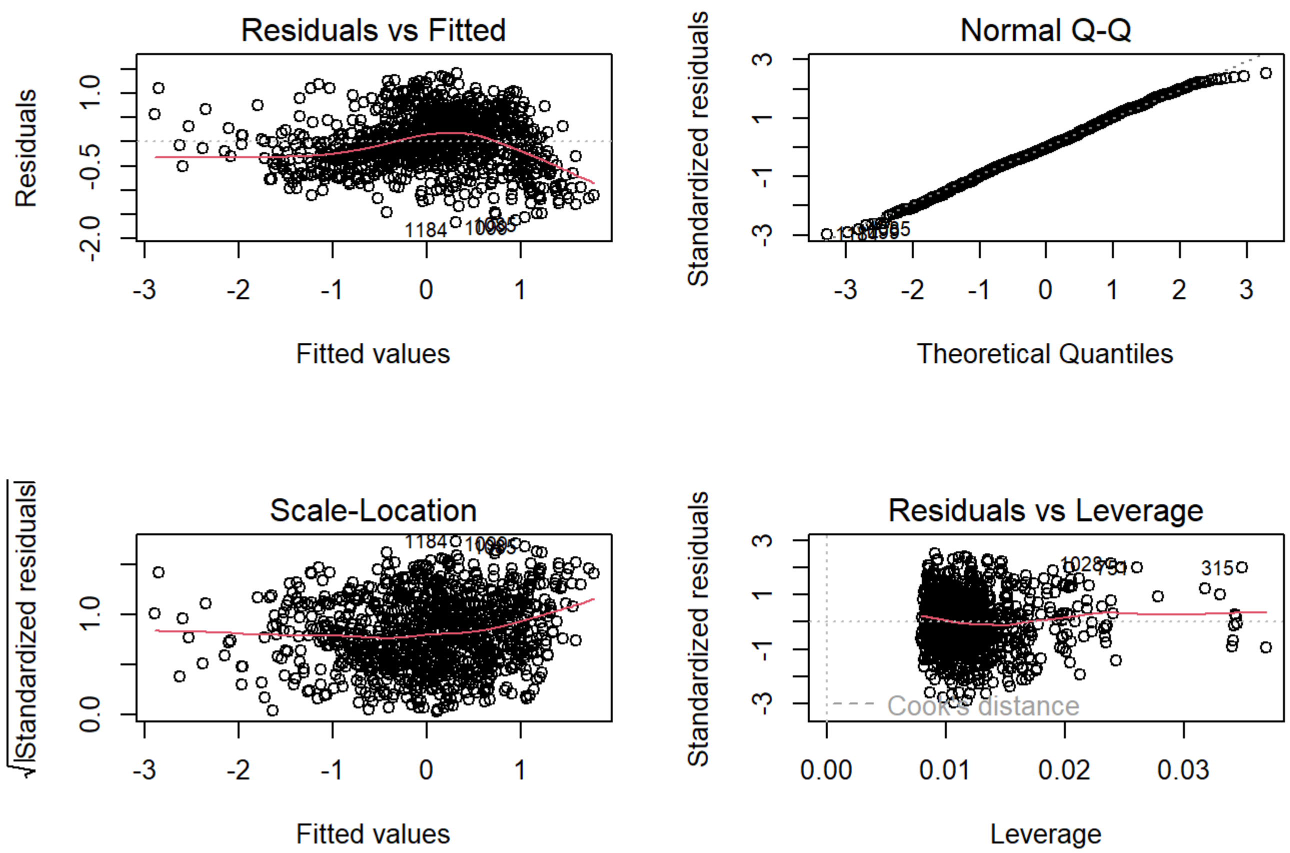
#Removing Outliers and leverage points

```
w <- abs(rstudent(mul_linear_fit)) < 3 & abs(cooks.distance(mul_linear_fit)) < 4/nrow(mul_linear
_fit$model)
LR_updated <- update(mul_linear_fit, weights=as.numeric(w))
summary(LR_updated)
```

```
##
## Call:
## lm(formula = V6 ~ ., data = train, weights = as.numeric(w))
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6699 -0.3261  0.0000  0.3426  1.4179
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           0.34635    0.06550   5.288 1.53e-07 ***
## V1                   -0.71224    0.02217 -32.126  < 2e-16 ***
## V2                   -0.32153    0.04032  -7.975 4.26e-15 ***
## V3-0.916694132600736 -0.14435    0.06869  -2.101   0.0359 *
## V3-0.373615188454538 -0.37045    0.07738  -4.787 1.95e-06 ***
## V30.169463755691661  -0.86097    0.07624 -11.294  < 2e-16 ***
## V30.98408217191096   -1.09107    0.07578 -14.398  < 2e-16 ***
## V31.79870058813026   -1.38627    0.08466 -16.375  < 2e-16 ***
## V4-0.723104159821969  0.12085    0.05252   2.301   0.0216 *
## V40.29790785680309    0.31951    0.05959   5.362 1.03e-07 ***
## V41.31249841420409    0.53324    0.05473   9.743  < 2e-16 ***
## V5                   -0.31056    0.03690  -8.415  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5639 on 970 degrees of freedom
## Multiple R-squared:  0.643,  Adjusted R-squared:  0.639
## F-statistic: 158.9 on 11 and 970 DF,  p-value: < 2.2e-16
```

After Removing Outliers and leverage points in training model : RSE = 0.5577; R2 = 0.6405(64%)

```
par(mfrow = c(2,2))
plot(LR_updated)
```




```
test_result = predict(mul_linear_fit,train)
predictions = test_result
actual = train$V6
mean((predictions - actual)^2)
```

```
## [1] 0.4802573
```

```
test_result = predict(mul_linear_fit,test)
predictions = test_result
actual = test$V6
mean((predictions - actual)^2)
```

```
## [1] 0.4678734
```

Mul_linear regression has a test error = 0.476

#LOOCV

```
glm.fit <- glm (V6 ~ ., data = train)
cv.err <- cv.glm(train , glm.fit)
cv.err$delta
```

```
## [1] 0.4920446 0.4920389
```

```
test_result = predict(glm.fit,test)
predictions = test_result
actual = test$V6
mean((predictions - actual)^2)
```

```
## [1] 0.4678734
```

#K=5 fold

```
glm.fit <- glm (V6 ~ ., data = train)
cv.err <- cv.glm(train , glm.fit, K = 5)
cv.err$delta
```

```
## [1] 0.5002621 0.4979966
```

#K=10 fold

```
glm.fit <- glm (V6 ~ ., data = train)
cv.err <- cv.glm(train , glm.fit, K = 10)
cv.err$delta
```

```
## [1] 0.4935680 0.4928621
```

#k=5 fold

```
set.seed (1)
cv.error.10 <- rep (0, 10)
for (i in 1:10) {
 glm.fit <- glm (V6 ~., data = train)
 cv.error.10[i] <- cv.glm(train , glm.fit , K = 5)$delta[1]
 }
cv.error.10
```

```
##  [1] 0.4949041 0.4904444 0.4902088 0.4921237 0.4910182 0.4902753 0.4917113
##  [8] 0.4964432 0.4943708 0.4894126
```

#k = 10 fold

```
set.seed (1)
cv.error.10 <- rep (0, 10)
for (i in 1:10) {
 glm.fit <- glm (V6 ~ ., data = train)
 cv.error.10[i] <- cv.glm(train , glm.fit , K = 10)$delta[1]
 }
cv.error.10
```

```
##  [1] 0.4926319 0.4944141 0.4907672 0.4913470 0.4925734 0.4934037 0.4921190
##  [8] 0.4932529 0.4918210 0.4988043
```

```
cv.error <- rep (0, 4)
for (i in 1:4) {
 glm.fit <- glm (V6 ~ poly (V2 , i), data = train)
 cv.error[i] <- cv.glm (train , glm.fit)$delta[1]
}
cv.error
```

```
## [1] 0.9526533 0.9538332 0.9536302 0.9462182
```

```
cv.error <- rep (0, 4)
for (i in 1:4) {
 glm.fit <- glm (V6 ~ poly (V3 , i), data = train)
 cv.error[i] <- cv.glm (train , glm.fit)$delta[1]
}
cv.error
```

```
## [1] 0.9208559 0.9215994 0.9227461 0.9218630
```

```
cv.error <- rep (0, 3)
for (i in 1:3) {
 glm.fit <- glm (V6 ~ poly (V4 , i), data = train)
 cv.error[i] <- cv.glm (train , glm.fit)$delta[1]
}
cv.error
```

```
## [1] 0.9590725 0.9601555 0.9550122
```

```
cv.error <- rep (0, 4)
for (i in 1:4) {
 glm.fit <- glm (V6 ~ poly (V5 , i), data = train)
 cv.error[i] <- cv.glm(train , glm.fit)$delta[1]
}
cv.error
```

```
## [1] 0.8910039 0.8896028 0.8748826 0.8730483
```

####Non-Linear Regression

From cor values only V1, V3, V5 have strong impact so only considered them

```
mul_non_linear = lm(V6 ~ V4+V2+V3+poly(V1,2,raw=T)+V5, data=train)
summary(mul_non_linear)
```

```
##
## Call:
## lm(formula = V6 ~ V4 + V2 + V3 + poly(V1, 2, raw = T) + V5, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.58626 -0.38256 -0.00634  0.45168  2.04542
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            0.23083    0.07206   3.203 0.001401 **
## V4-0.723104159821969   0.09585    0.06129   1.564 0.118140
## V40.29790785680309     0.30636    0.06921   4.426 1.06e-05 ***
## V41.31249841420409     0.53573    0.06398   8.373  < 2e-16 ***
## V2                    -0.34938    0.04440  -7.869 8.92e-15 ***
## V3-0.916694132600736  -0.13216    0.07710  -1.714 0.086798 .
## V3-0.373615188454538  -0.30072    0.08590  -3.501 0.000484 ***
## V30.169463755691661   -0.82716    0.08495  -9.737  < 2e-16 ***
## V30.98408217191096    -1.00875    0.08317 -12.129  < 2e-16 ***
## V31.79870058813026    -1.35307    0.09410 -14.379  < 2e-16 ***
## poly(V1, 2, raw = T)1 -0.76712    0.03432 -22.353  < 2e-16 ***
## poly(V1, 2, raw = T)2  0.09087    0.01231   7.383 3.18e-13 ***
## V5                    -0.26283    0.04047  -6.495 1.28e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6797 on 1039 degrees of freedom
## Multiple R-squared:   0.53,  Adjusted R-squared:  0.5245
## F-statistic: 97.62 on 12 and 1039 DF,  p-value: < 2.2e-16
```

Non-Linear Model:RSE = 0.6797, R2 = 0.53

```
test_result_nl = predict(mul_non_linear,test)
predictions = test_result_nl
actual = test$V6
mean((predictions - actual)^2)
```

```
## [1] 0.4367219
```

non_linear_multiple_test error = 0.55

LOOCV

```
library (boot)
glm.fit <- glm (V6~V4+V2+V3+poly(V1,2,raw=T)+V5, data=train)
cv.err <- cv.glm(train , glm.fit)
cv.err$delta
```

```
## [1] 0.4679065 0.4679009
```

LOOCV error = 0.503

#k=5 fold

```
set.seed (1)
glm.fit <- glm (V6~V4+V2+V3+poly(V1,2,raw=T)+V5, data=train)
cv.error <- cv.glm(train , glm.fit , K = 5)$delta[1]
cv.error
```

```
## [1] 0.4688194
```

#k = 10 fold

```
set.seed (1)
glm.fit <- glm (V6~V4+V2+V3+poly(V1,2,raw=T)+V5, data=train)
cv.error <- cv.glm(train , glm.fit , K = 10)$delta[1]
cv.error
```

```
## [1] 0.4675715
```

#Ridge_regression

```
x <- model.matrix (V6 ~ ., data = scale_data)[,-1]
y <- scale_data$V6
```

```
set.seed (1)
train <- sample (1:nrow(x),nrow(x)/2)
test <- (-train)
y.test <- y[test]
```

```
ridge.mod <- glmnet(x[train,], y[train], alpha = 0, thresh=1e-12)
ridge.pred <- predict(ridge.mod,s=4,newx = x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 0.8443143
```

```
mean((mean(y[train])- y.test)^2)
```

```
## [1] 0.9766397
```

```
ridge.pred <- predict (ridge.mod , s = 1e10 , newx = x[test , ])
mean ((ridge.pred - y.test)^2)
```

```
## [1] 0.9766397
```

```
ridge.pred <- predict (ridge.mod , s = 0, newx = x[test , ],exact = T, x = x[train , ], y = y[tr
ain])
mean((ridge.pred - y.test)^2)
```

```
## [1] 0.4791767
```

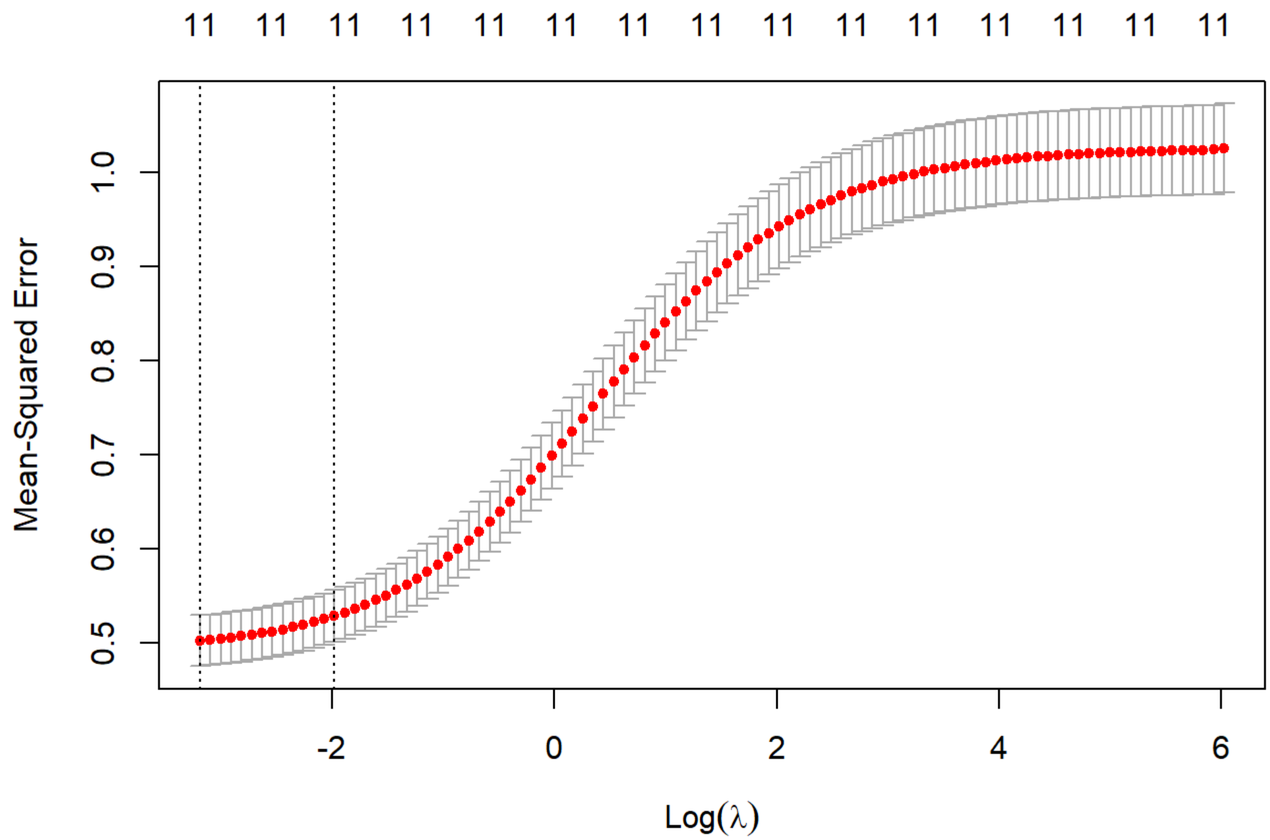
```
lm(y ~ x, subset = train)
```

```
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
##          (Intercept)                   xV1                   xV2
##              0.46810              -0.56481              -0.43500
## xV3-0.916694132600736  xV3-0.373615188454538   xV30.169463755691661
##             -0.28435              -0.47893              -1.05494
##   xV30.98408217191096    xV31.79870058813026  xV4-0.723104159821969
##             -1.17991              -1.54878               0.08393
##   xV40.29790785680309    xV41.31249841420409                   xV5
##              0.36617               0.56159              -0.20347
```

```
predict (ridge.mod , s = 0, exact = T, type = "coefficients",
x = x[train , ], y = y[train])[1:6, ]
```

```
##          (Intercept)                    V1                    V2
##            0.4680943            -0.5648091            -0.4349985
## V3-0.916694132600736 V3-0.373615188454538  V30.169463755691661
##           -0.2843451            -0.4789175            -1.0549306
```

```
library(glmnet)
set.seed (1)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 0)
plot (cv.out)
```




```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 0.04123489
```

Best lambda = 0.0412

```
ridge.pred <- predict (ridge.mod , s = bestlam ,
newx = x[test , ])
```

```
mean((ridge.pred - y.test)^2)
```

```
## [1] 0.4826938
```

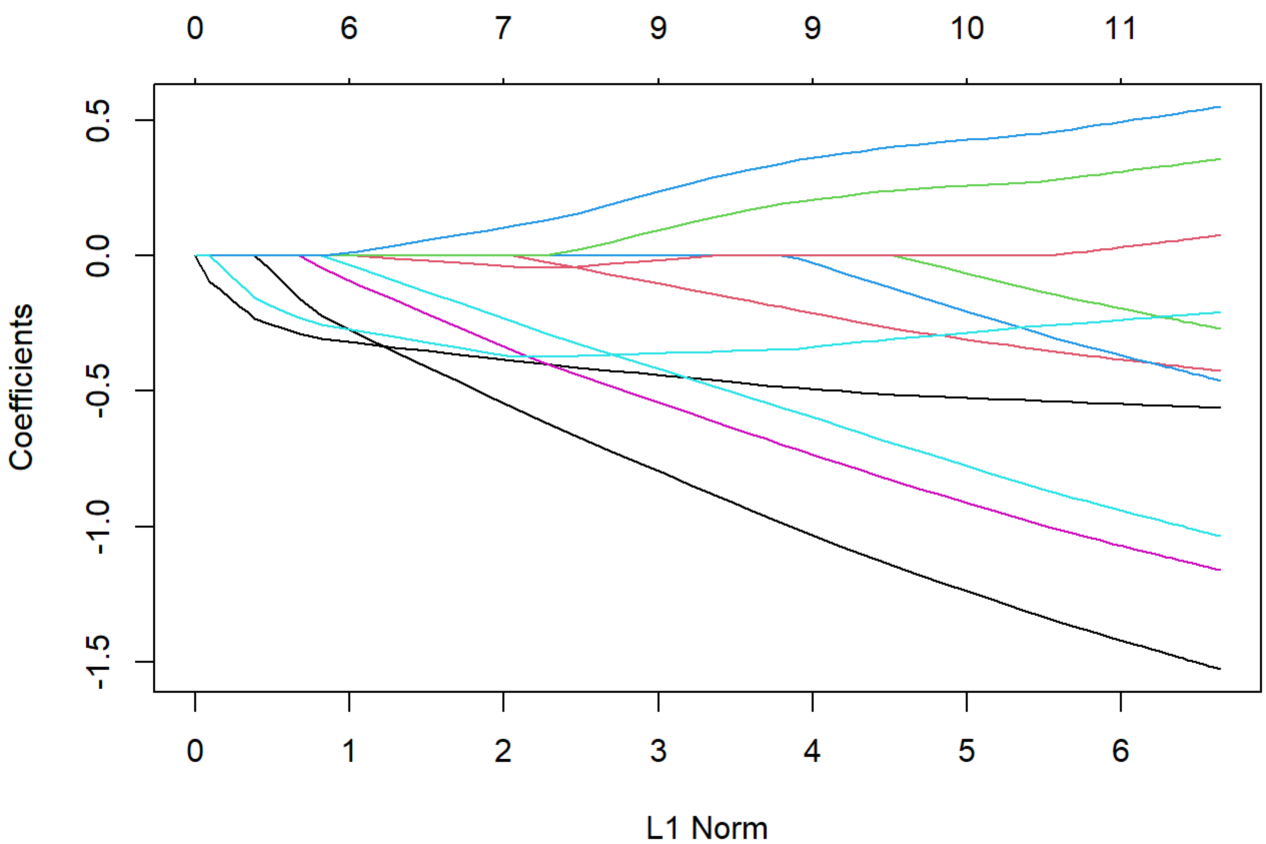
Ridge Regression has a mean = 0.48

```
out <- glmnet (x, y, alpha = 0)
predict (out , type = "coefficients", s = bestlam)[1:6, ]
```

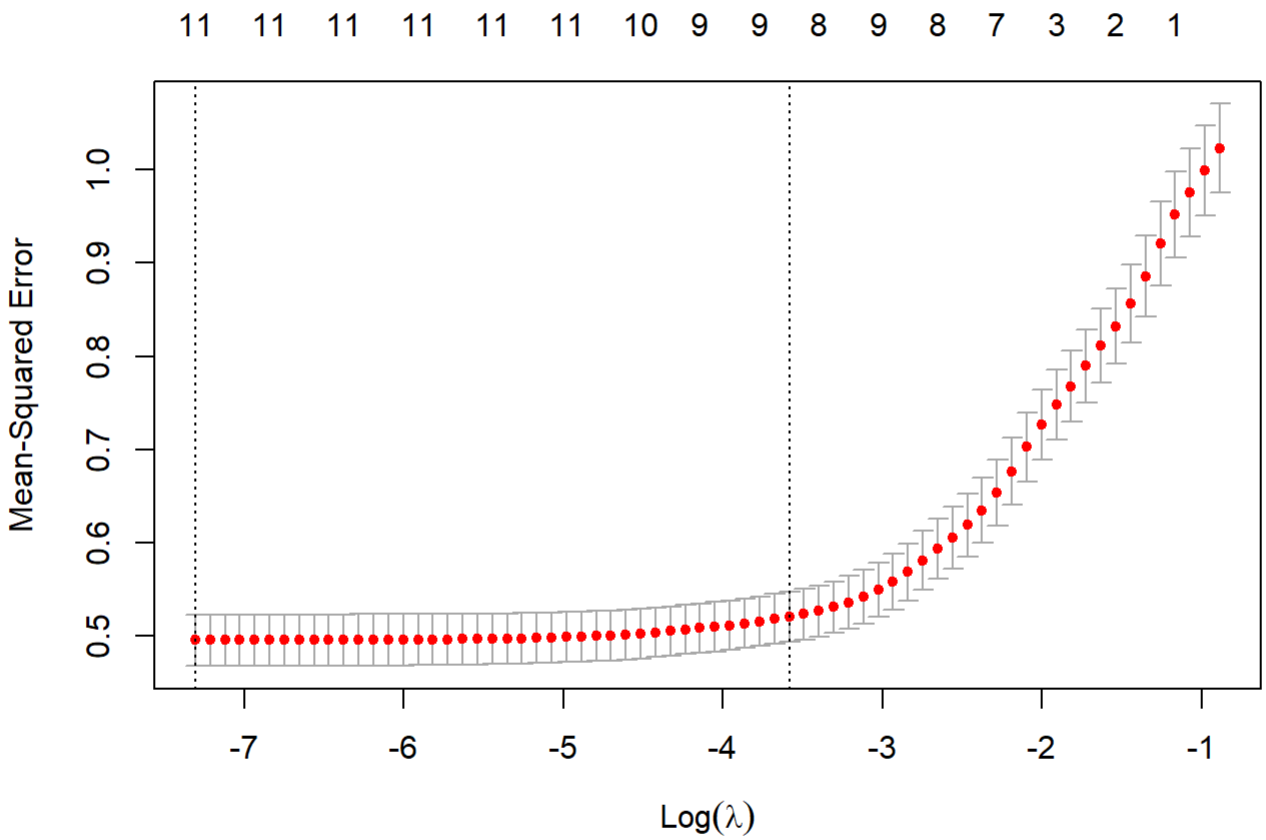
```
##          (Intercept)                   V1                   V2
##           0.25630703          -0.54116761          -0.28433970
## V3-0.916694132600736 V3-0.373615188454538  V30.169463755691661
##          -0.07598015          -0.20464760          -0.70431113
```

#LASSO_Regression

```
lasso.mod <- glmnet (x[train , ], y[train], alpha = 1)
plot (lasso.mod)
```




```
set.seed (1)
cv.out <- cv.glmnet (x[train , ], y[train], alpha = 1)
plot (cv.out)
```




```
bestlam <- cv.out$lambda.min
lasso.pred <- predict (lasso.mod , s = bestlam ,
newx = x[test , ])
mean ((lasso.pred - y.test)^2)
```

```
## [1] 0.4789523
```

LAsso Regression has a mean = 0.481

```
out <- glmnet (x, y, alpha = 1)
lasso.coef <- predict (out , type = "coefficients",
s = bestlam)[1:6, ]
lasso.coef
```

```
##         (Intercept)                    V1                    V2
##           0.3765570            -0.5888581            -0.3931809
## V3-0.916694132600736 V3-0.373615188454538  V30.169463755691661
##          -0.2168736            -0.4069441            -0.9285473
```

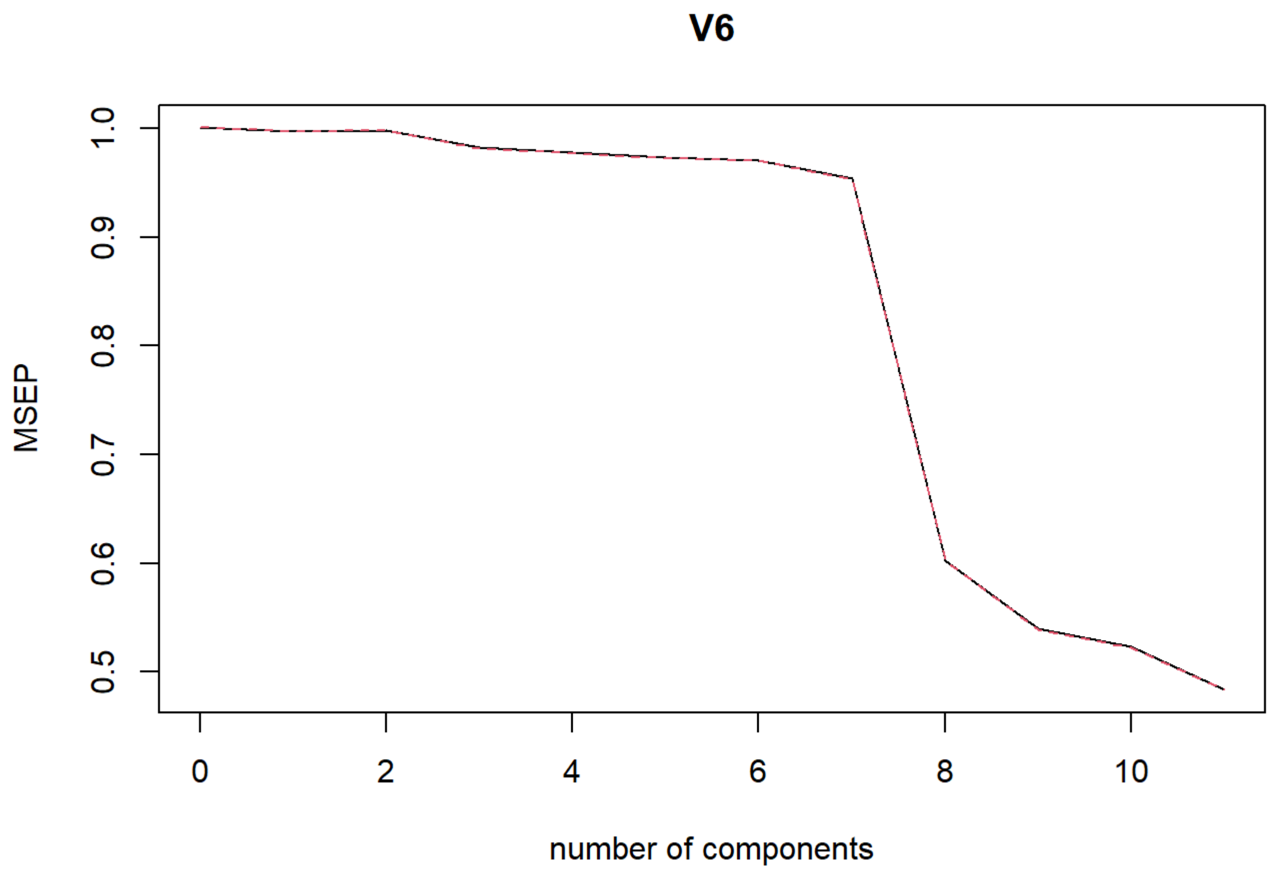
PCR model

```r
#install.packages("pls")
set.seed (2)
pcr.fit <- pcr(V6 ~ ., data = scale_data , scale = TRUE ,validation = "CV")
```

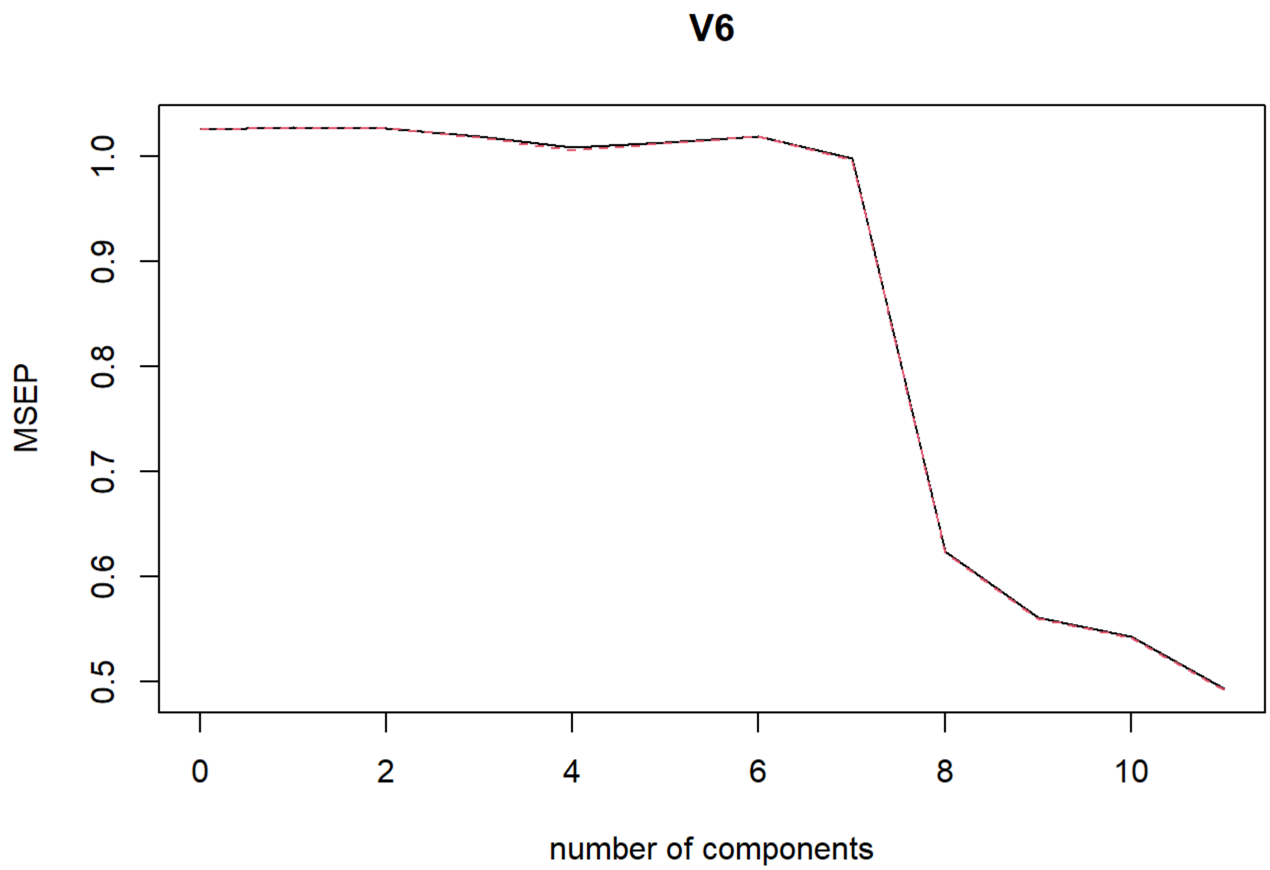
```r
summary(pcr.fit)
```

```
## Data:    X dimension: 1503 11
##  Y dimension: 1503 1
## Fit method: svdpc
## Number of components considered: 11
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV               1   0.9989   0.9992   0.9911   0.9888   0.9865   0.9853
## adjCV            1   0.9988   0.9991   0.9906   0.9884   0.9863   0.9851
##        7 comps  8 comps  9 comps  10 comps  11 comps
## CV      0.9766   0.7763   0.7345    0.7230    0.6956
## adjCV   0.9762   0.7759   0.7342    0.7227    0.6952
##
## TRAINING: % variance explained
##     1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X   20.2815  33.7804   45.709   56.857   67.810   78.411   87.120    93.78
## V6   0.4793   0.7626    3.117    3.864    4.278    4.575    6.393    40.66
##     9 comps  10 comps  11 comps
## X     96.94     99.28    100.00
## V6    46.98     48.63     52.49
```

```r
validationplot (pcr.fit , val.type = "MSEP")
```




```
set.seed (1)
pcr.fit <- pcr(V6 ~., data = scale_data , subset = train ,scale = TRUE , validation = "CV")
validationplot (pcr.fit , val.type = "MSEP")
```




```
pcr.pred <- predict (pcr.fit , x[test , ], ncomp = 5)
mean((pcr.pred - y.test)^2)
```

```
## [1] 0.9395918
```

PCR Test error = 0.479

```
pcr.fit <- pcr(y ~ x, scale = TRUE , ncomp = 5)
summary (pcr.fit)
```

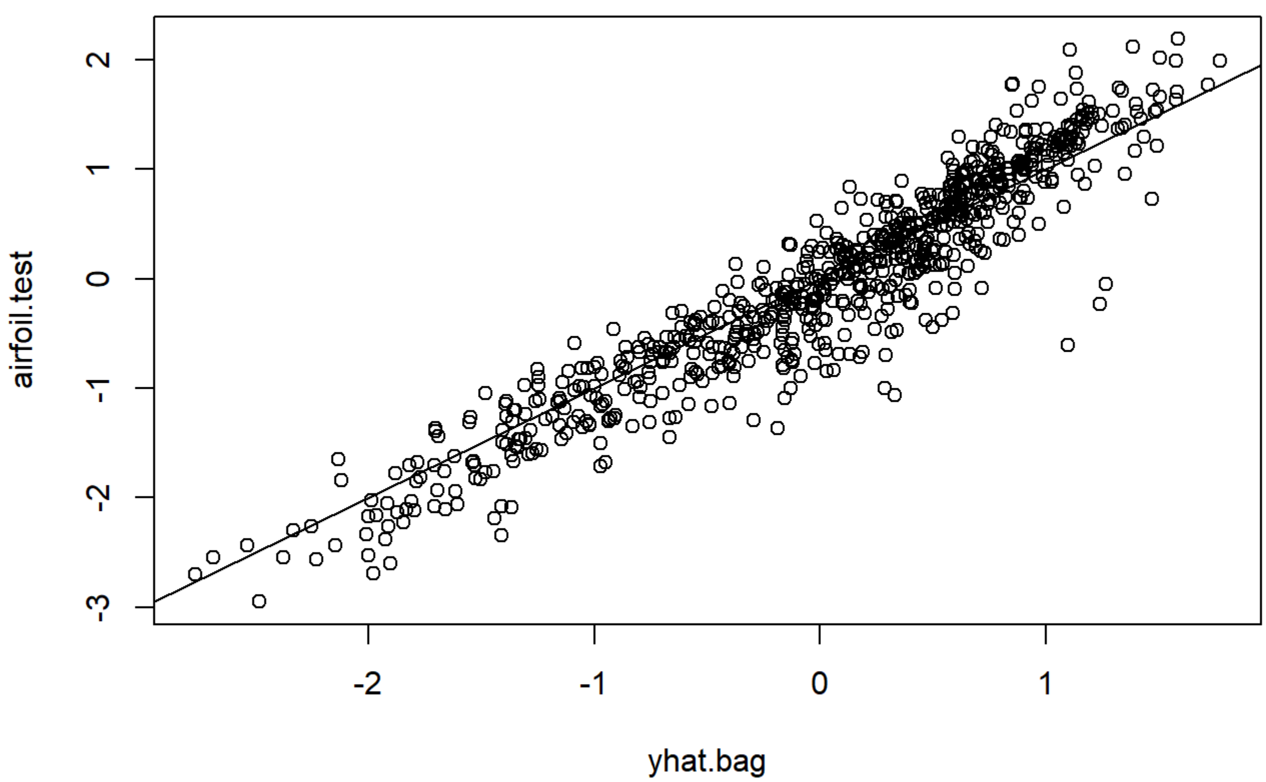
```
## Data:    X dimension: 1503 11
##  Y dimension: 1503 1
## Fit method: svdpc
## Number of components considered: 5
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps
## X  20.2815  33.7804   45.709   56.857   67.810
## y   0.4793   0.7626    3.117    3.864    4.278
```

#Bagging and Rando Forests

```r
library (randomForest)
set.seed (1)
bag.airfoil <- randomForest(V6 ~ .,data = scale_data,subset = train,mtry =5, importance = TRUE)
bag.airfoil
```

```
##
## Call:
##  randomForest(formula = V6 ~ ., data = scale_data, mtry = 5, importance = TRUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 0.1071774
##                     % Var explained: 89.52
```

```r
yhat.bag <-predict(bag.airfoil,newdata = scale_data[-train , ])
airfoil.test <- scale_data[-train , "V6"]
plot(yhat.bag,airfoil.test)
abline(0, 1)
```



```
mean((yhat.bag - airfoil.test)^2)
```

```
## [1] 0.1176819
```

MSE Test = 0.1101373 for mtry = 5

```
importance(bag.airfoil)
```

```
##       %IncMSE IncNodePurity
## V1 199.31257     288.95331
## V2  37.23624      37.28976
## V3  84.73663      91.35525
## V4  34.54984      29.98994
## V5 142.59746     299.71847
```

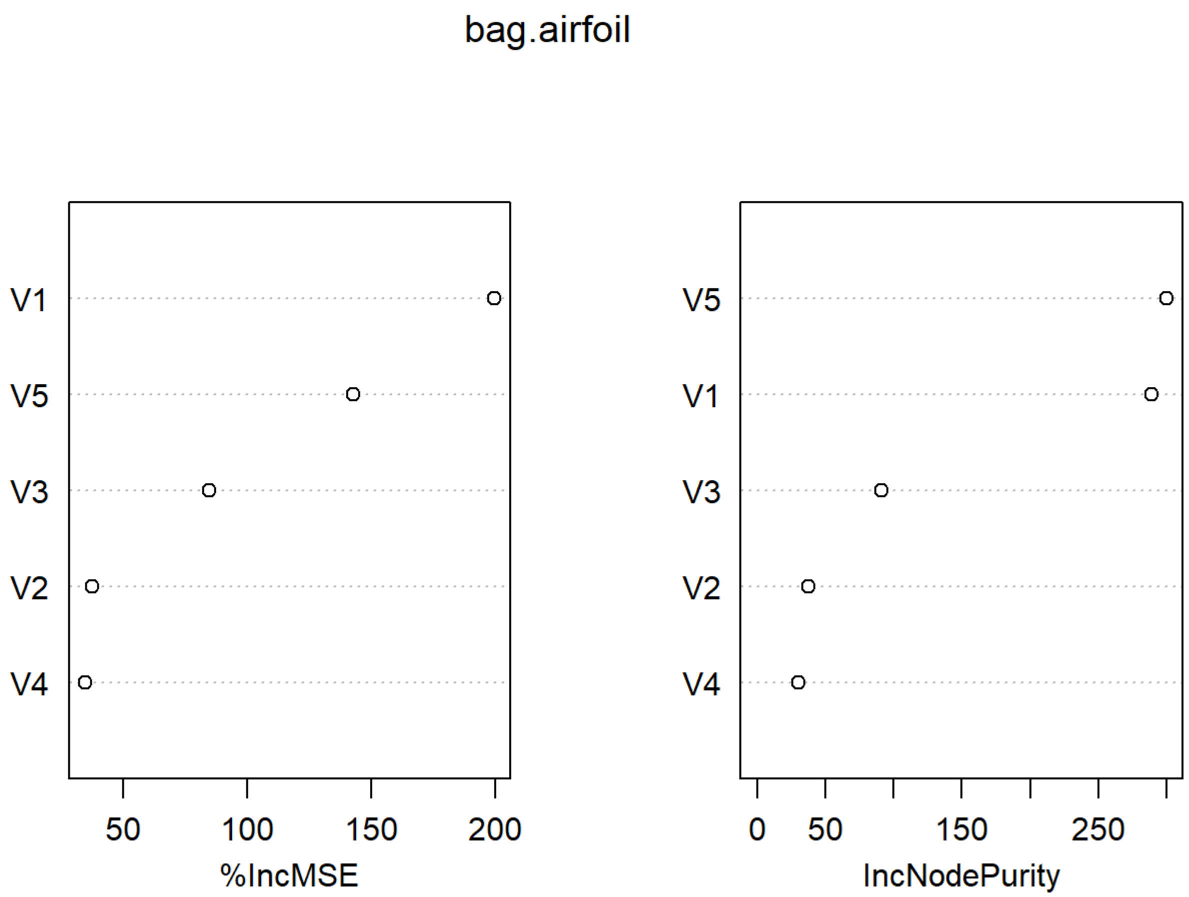
Every variable is important if removed one variable also test error increases.

```
bag.airfoil_2 <- randomForest(V6 ~ V1+V2+V3+V5,data = scale_data,subset = train,mtry =4, importa
nce = TRUE)
yhat.bag <-predict(bag.airfoil_2,newdata = scale_data[-train , ])
airfoil.test <- scale_data[-train , "V6"]
mean((yhat.bag - airfoil.test)^2)
```

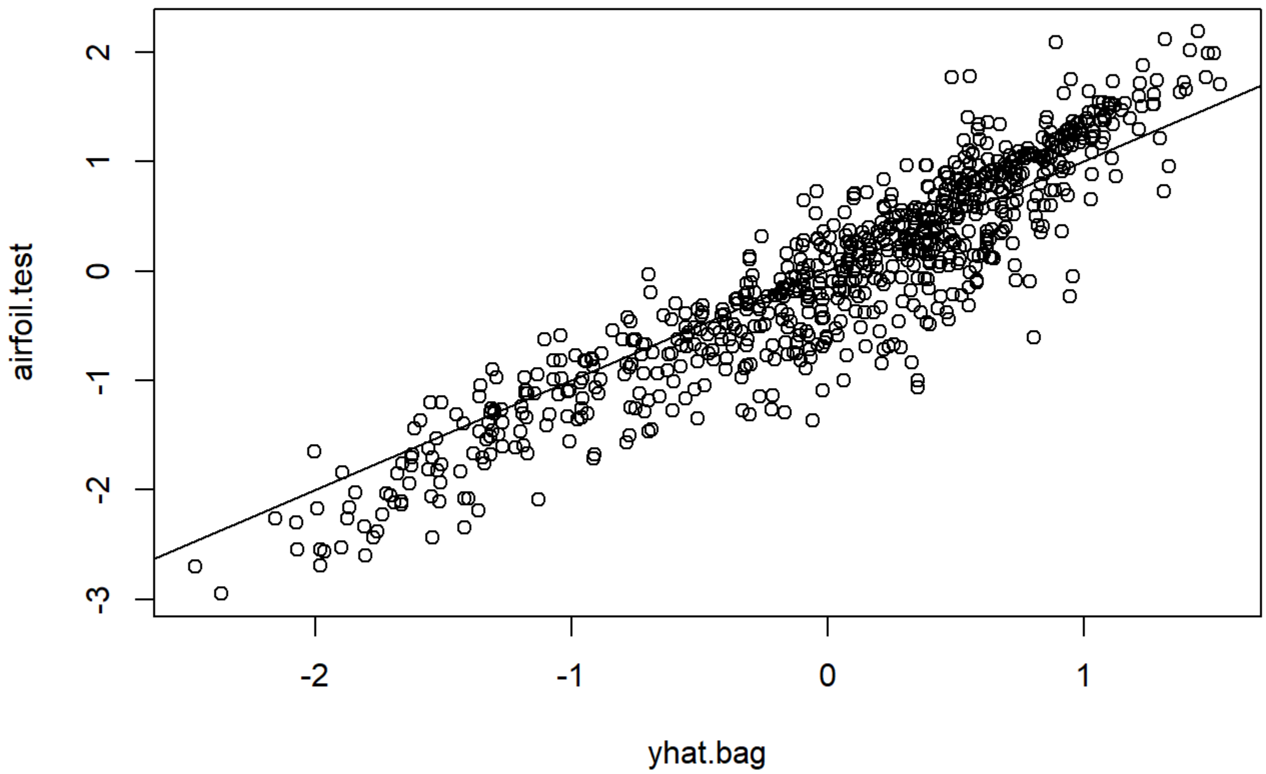
```
## [1] 0.1234491
```

See test error increased.

```
varImpPlot(bag.airfoil)
```




```
bag.airfoil_1 <- randomForest(V6 ~ .,data = scale_data,subset = train,mtry =2, importance = TRU
E)
yhat.bag <-predict(bag.airfoil_1,newdata = scale_data[-train , ])
airfoil.test <- scale_data[-train , "V6"]
plot(yhat.bag,airfoil.test)
abline(0, 1)
```




```
mean((yhat.bag - airfoil.test)^2)
```

```
## [1] 0.1621426
```

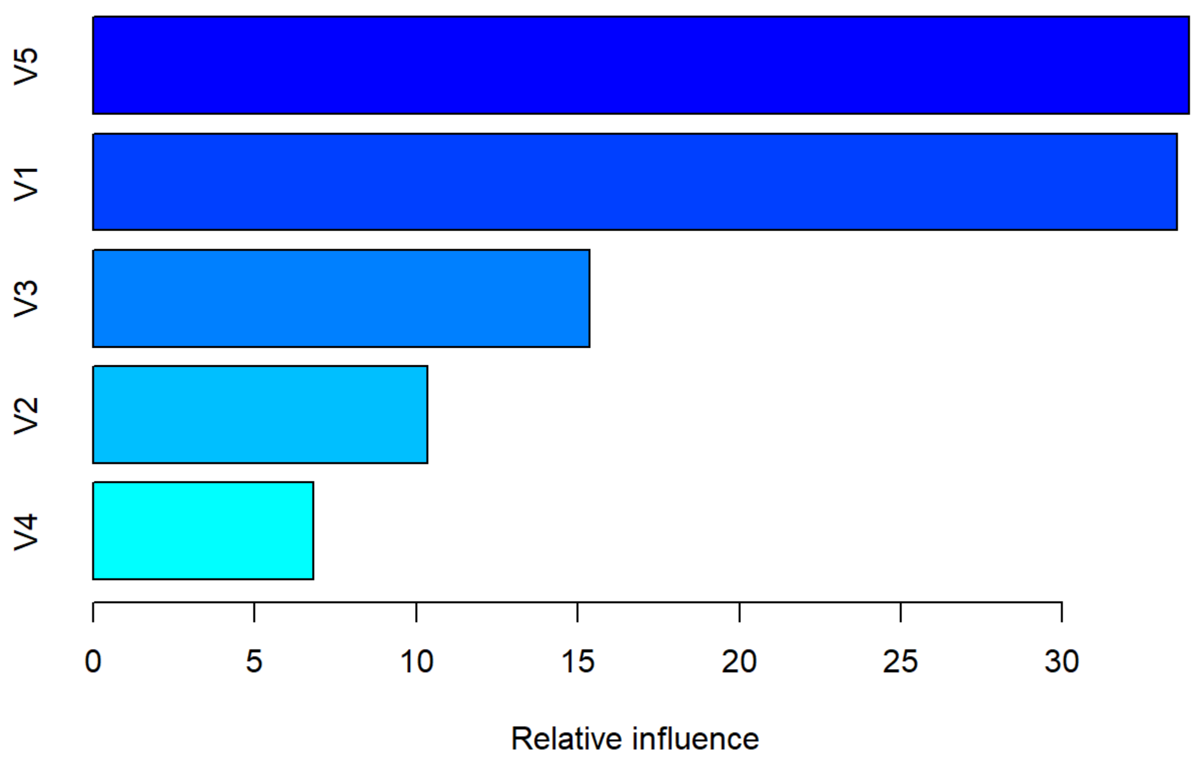
MSE Test = 0.1544284 for mtry = 2 and observed lowest MSE test for mtry=5

```
bag.airfoil_3 <- randomForest(V6 ~ .,data = scale_data,subset = train,mtry =5,ntree = 50)
yhat.bag <-predict(bag.airfoil_3,newdata = scale_data[-train , ])
airfoil.test <- scale_data[-train , "V6"]
mean((yhat.bag - airfoil.test)^2)
```

```
## [1] 0.1241689
```

#BOOSTING

```
set.seed (1)
boost.airfoil <- gbm(V6 ~ ., data = scale_data[train , ],distribution = "gaussian", n.trees = 50
00,interaction.depth = 4)
summary (boost.airfoil)
```




```
##    var   rel.inf
## V5  V5 33.916823
## V1  V1 33.556623
## V3  V3 15.379781
## V2  V2 10.335428
## V4  V4  6.811345
```

```
yhat.boost <- predict (boost.airfoil,newdata = scale_data[-train , ],n.trees = 5000)
mean((yhat.boost - airfoil.test)^2)
```

```
## [1] 0.09330604
```

Boosting has an test error = 0.09058552.

#Bayesian Additive Regression Trees

```
library(BART)
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## Loading required package: nnet
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:boot':
##
##     aml
```

```
x <- scale_data[, 1:5]
y <- scale_data[, "V6"]
xtrain <- x[train , ]
ytrain <- y[train]
xtest <- x[-train , ]
ytest <- y[-train]
set.seed (1)
bartfit <- gbart (xtrain , ytrain , x.test = xtest)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 751, 13, 752
## y1,yn: -0.114136, 0.078801
## x1,x[n*p]: 4.159656, 1.075161
## xp1,xp[np*p]: -0.661802, 3.171717
## *****Number of Trees: 200
## *****Number of Cut Points: 20 ... 100
## *****burn,nd,thin: 100,1000,1
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.0963672,3,0.0941911,0.0231117
## *****sigma: 0.695377
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,13,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 5s
## trcnt,tecnt: 1000,1000
```

```
yhat.bart <- bartfit$yhat.test.mean
mean((ytest - yhat.bart)^2)
```

```
## [1] 0.1222752
```