# ML models for quantitative & categorical target variables

Quy Do

Nov 2021

*The purpose of this project is using R programming language to showcase the data analytical techniques when dealing with (i) quantitative target variable and (ii) categorical target variable. We will investigate each dataset for each case, determine the most suited machine learning model to predict the target variable and predict unseen observations.*

```
library(readr)
library(glmnet)
library(splines)
library(GGally)
library(dplyr)
```

## I. Dataset with quantitative target variable

### 1.1. Import data

Data is firstly imported to R as **df1**. This is a dataset including 5000 observations, in which, x1, x2 and x5 are continuous variables where x3 and x4 are categorical variables.

```
df1 = read.csv("df1_quant_y.csv")
head(df1)
```

```
##            y         x1        x2 x3 x4         x5
## 1  3.9827442 -0.6140529  2.222503  4  1  2.4466606
## 2 -0.7676510 -0.9491327  7.588784  2  0 -0.2624016
## 3  6.6056254  0.2934879  9.684209  1  1  4.4420065
## 4  9.6063076  0.3912818  3.144804  0  0  0.8599485
## 5 18.6112095  1.0206324 16.045787  1  0  1.6897057
## 6  0.6984214 -1.2677542 12.195228  1  0  1.7593135
```

```
nrow(df1) #number of observations
```

```
## [1] 5000
```

```
summary(df1)
```

```
##        y                  x1                  x2                 x3
##  Min.   :-14.6683   Min.   :-4.33578   Min.   : 0.06551   Min.   :0.000
##  1st Qu.:  0.9293   1st Qu.:-0.70096   1st Qu.: 2.64880   1st Qu.:1.000
```

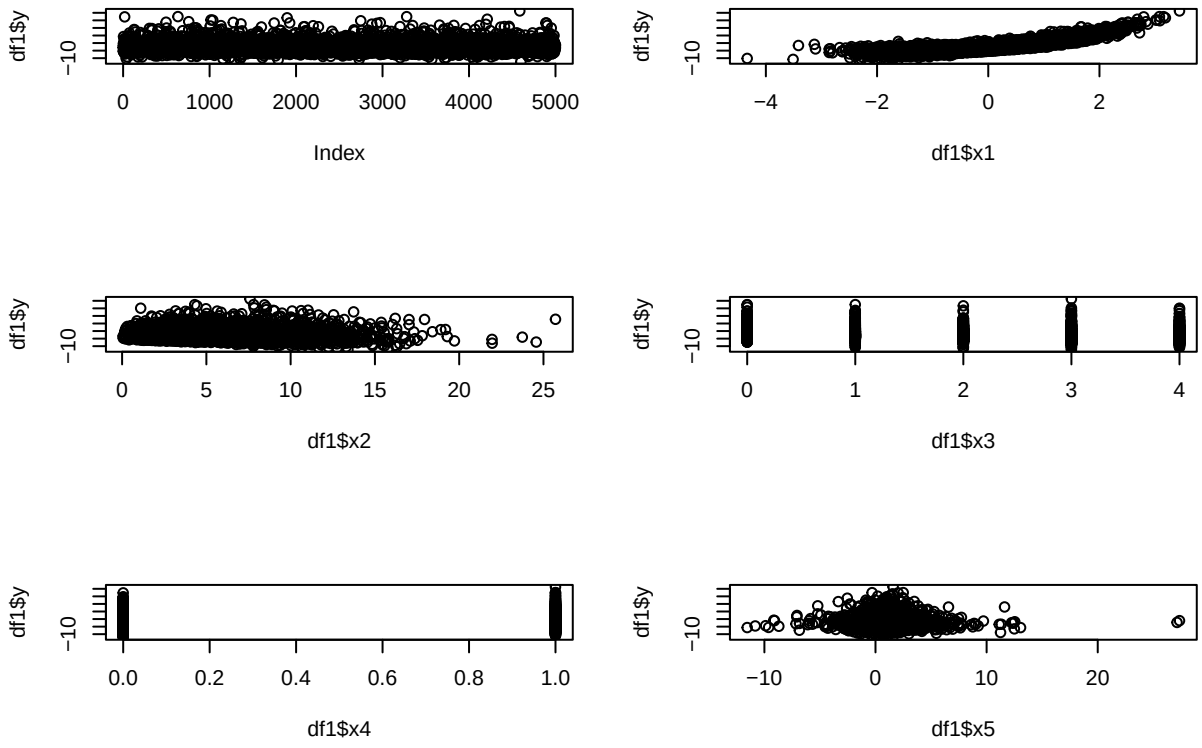
```
##  Median :  4.1427   Median :-0.02612   Median : 4.37017   Median :2.000
##  Mean   :  5.3276   Mean   :-0.02204   Mean   : 5.03379   Mean   :2.005
##  3rd Qu.:  8.1659   3rd Qu.: 0.64284   3rd Qu.: 6.70540   3rd Qu.:3.000
##  Max.   : 52.4708   Max.   : 3.43926   Max.   :25.71290   Max.   :4.000
##        x4                 x5
##  Min.   :0.0000   Min.   :-11.5470
##  1st Qu.:0.0000   1st Qu.:  0.2229
##  Median :0.0000   Median :  1.0077
##  Mean   :0.4936   Mean   :  1.0264
##  3rd Qu.:1.0000   3rd Qu.:  1.7688
##  Max.   :1.0000   Max.   : 27.3584
```

From the summary of df1, value of y is ranging from [-14.6683, 52.4708] with its mean is 5.3276.

### 1.2. Graphically examination of y and x:

Before conducting any data manipulation, we want to observe the relationships between target variable y and the 5 explanatory variables singly.

```
par(mfrow = c(3,2))
plot(df1$y)+
plot(df1$x1, df1$y) +
plot(df1$x2, df1$y) +
plot(df1$x3, df1$y) +
plot(df1$x4, df1$y) +
plot(df1$x5, df1$y)
```



```
## integer(0)
```

- The plot between $y$ and $x_1$ shows a positive relationship. $x_1$ might be in the polynomial regression form.
- The plots between $y$ and $x_2$, $x_5$ do not show any clear trends.

### 1.3. Data manipulation

To build a ML model, we want to split **df1** into training and test data sets in which 80% observations are for training and 20% for test data sets.

```
# Split the dataset up into training and test sets
df1_train = df1[1:4000, ]
df1_test = df1[4001:nrow(df1), ]
```

Also, we want to create the column of ones having the same length as training and test data sets. This will be useful for matrix construction in the later stage of model building.

```
ones_train_df1 = rep(1, nrow(df1_train))
ones_test_df1 = rep(1, nrow(df1_test))
```

### 1.4. Model selection

In the process of selecting the optimal model, we select the model that minimizes the expected loss by taking advantage of forward/backward step-wise selection method to pick the model giving the best AIC. In other words, we want to identify the important features contributing to the accuracy of the the ML model.

Alternatively, we can try to apply penalty functions in the complex model using Lasso and Ridge regression. This regression will gradually reducing the parameter estimates towards zero, so that the model complexity will be reduced.

#### 1. Step-wise regression

Forward and backward step-wise regressions are used as below, in which both approaches lead to the same result. Particularly:

```
# Perform forward step-wise regression on the training set
model1 = lm(y ~ 1, data=df1_train)
step_formula = as.formula("~x1+x2+x3+x4+x5")
model1step = step(model1, scope=step_formula, direction="forward")
```

```
## Start:  AIC=15294.62
## y ~ 1
##
##        Df Sum of Sq    RSS   AIC
## + x1    1    125537  57456 10663
## + x3    1     13621 169372 14987
## + x4    1      1027 181966 15274
## + x2    1       103 182891 15294
## <none>              182993 15295
## + x5    1         4 182990 15296
##
## Step:  AIC=10662.9
```

```
## y ~ x1
##
##        Df Sum of Sq   RSS     AIC
## + x3    1   12586.6 44869  9675.8
## + x4    1     901.9 56554 10601.6
## <none>              57456 10662.9
## + x2    1      12.9 57443 10664.0
## + x5    1       2.5 57454 10664.7
##
## Step:  AIC=9675.84
## y ~ x1 + x3
##
##        Df Sum of Sq   RSS    AIC
## + x4    1    992.32 43877 9588.4
## <none>              44869 9675.8
## + x2    1      2.02 44867 9677.7
## + x5    1      0.07 44869 9677.8
##
## Step:  AIC=9588.39
## y ~ x1 + x3 + x4
##
##        Df Sum of Sq   RSS    AIC
## <none>              43877 9588.4
## + x2    1    6.8464 43870 9589.8
## + x5    1    0.4355 43877 9590.3
```

- Forward step-wise selection results a model $y \sim x_1 + x_3 + x_4$ with the lowest AIC $= 9588.39$ among others. In other words, $x_1, x_3, x_4$ are important variables and should be included in the optimal model.
- Similarly, in the backward step-wise selection, the fitted model is also the model $y \sim x_1 + x_3 + x_4$.

```
all_formula = as.formula("y~x1+x2+x3+x4+x5")
model1 = lm(all_formula, data=df1_train)
model2step = step(model1, scope=step_formula, direction="backward")
```

```
## Start:  AIC=9591.73
## y ~ x1 + x2 + x3 + x4 + x5
##
##        Df Sum of Sq    RSS     AIC
## - x5    1         0  43870  9589.8
## - x2    1         7  43877  9590.3
## <none>               43870  9591.7
## - x4    1       997  44867  9679.7
## - x3    1     12660  56530 10603.9
## - x1    1    124292 168162 14964.5
##
## Step:  AIC=9589.76
## y ~ x1 + x2 + x3 + x4
##
##        Df Sum of Sq    RSS     AIC
## - x2    1         7  43877  9588.4
## <none>               43870  9589.8
## - x4    1       997  44867  9677.7
```

```
## - x3    1     12661  56532 10602.0
## - x1    1    124292 168162 14962.5
##
## Step:  AIC=9588.39
## y ~ x1 + x3 + x4
##
##        Df Sum of Sq    RSS     AIC
## <none>               43877  9588.4
## - x4    1       992  44869  9675.8
## - x3    1     12677  56554 10601.6
## - x1    1    124368 168245 14962.5
```

**2. Basis expansion for continuous variable**:

In the above model, $x_1$ is a continuous variable. Based on the plot between $x_1$ and $y$, we should perform basic expansion on $x_1$ using splines.

We would try different models with different knots of the splines and different polynomial forms of linear, quadratic and cubic splines (i.e. polynomial splines with degree of 1,2 and 3 respectively). We will not pursue higher-order polynomials as they normally have no practical gain. Furthermore, among the 3 degrees of polynomial regression, we will only examine linear and cubic splines as they are more popular and flexible, while quadratic splines are rarely mentioned in different disciplines. Furthermore, cubic splines add extra smoothness and generality which worth for the optimal model despite having additional parameter in comparison to quadratic splines. In short:

- We will firstly examine the simple linear basic expansion of $x_1$ around 1 knot.
- Then, we will examine and compare the expected loss between the first model with other models built from cubic basic expansion with more than 1 knot.

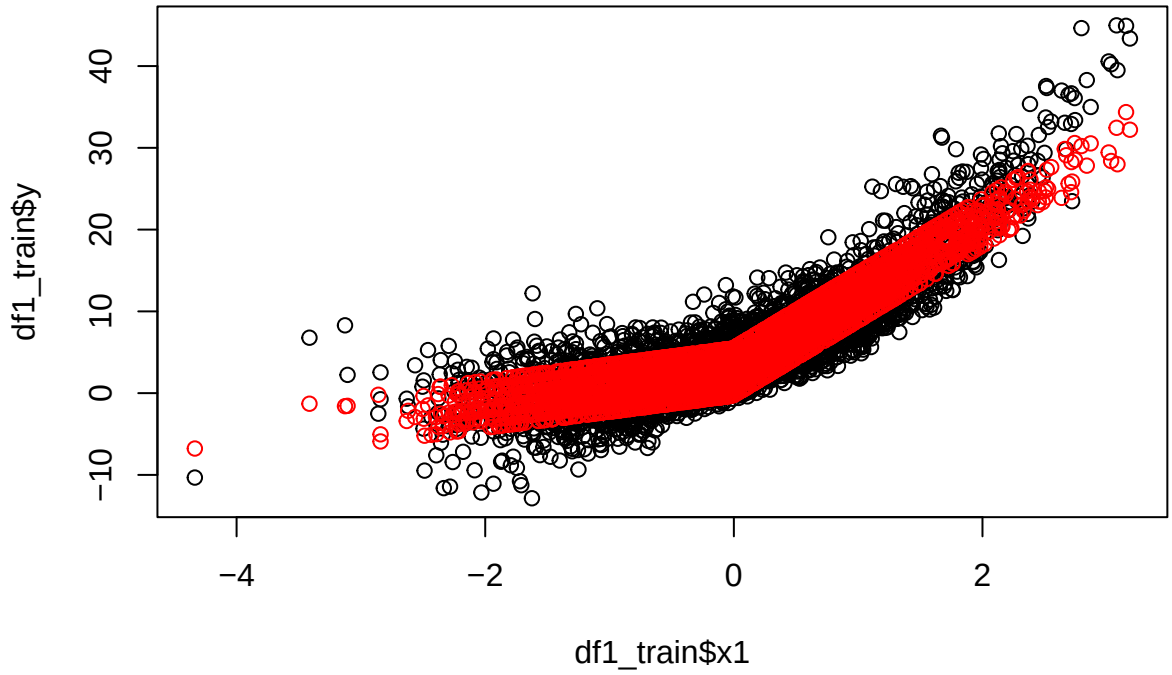
**Linear basis expansion of $x_1$ around 1 knot:**

```
#Create the knot at the 50% quantile of x1
knotvec1_df1 = quantile(df1_train$x1, probs=c(0.5))

#create the linear basis expansion of x1 around 1 knot
x1_lbe_train_df1 = bs(df1_train$x1, degree=1, knots=knotvec1_df1)

#create the matrix of independent variables
xall_lbe_train_df1 = cbind(ones_train_df1, x1_lbe_train_df1, df1_train$x3, df1_train$x4)

# Regress x-matrix on the linear basis expansion using glmnet
glm_lbe_df1 = glmnet(xall_lbe_train_df1, df1_train$y, family="gaussian")

# Compute the sample mean absolute loss on the test datset
x1_lbe_test_df1 = bs(df1_test$x1, degree=1, knots=knotvec1_df1)
xall_lbe_test_df1 = cbind(ones_test_df1, x1_lbe_test_df1, df1_test$x3, df1_test$x4)
yhat1_lbe_test_df1 = predict(glm_lbe_df1, newx=xall_lbe_test_df1, s=0)

loss1_lbe_df1 = abs(df1_test$y - yhat1_lbe_test_df1)
mean(loss1_lbe_df1)
```

```
## [1] 1.955761
```

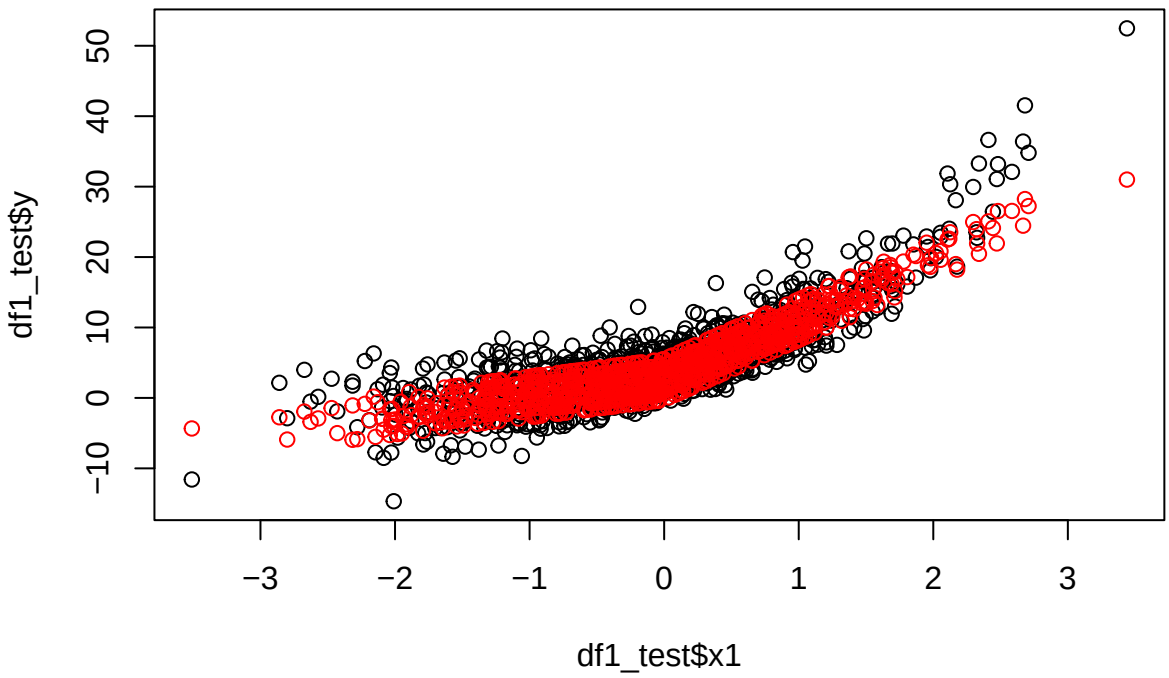
The mean of the absolute loss between the true value and the predicted value is 1.956. We can examine the plot of $x_1$ in training and test data sets where the predicted points are in red:

```
# Plot the training data, and then add the in-sample model predictions in red
plot(df1_train$x1, df1_train$y) +
points(df1_train$x1, predict(glm_lbe_df1, newx=xall_lbe_train_df1, s=0), col="red")
```




```
## integer(0)
```

```
# Plot on the test data to see the out-of-sample fit
plot(df1_test$x1, df1_test$y)+
points(df1_test$x1, yhat1_lbe_test_df1, col="red")
```




```
## integer(0)
```

With 1 knot in linear basic expansion, we can see a very sharp inflection point in the prediction comparing to the actual points. Thus, the non-linearity of the dataset is not well captured. We can improve the model by adding more knots as well as comparing with cubic regression model. As a result, below is the summary table of the absolute expected loss on the out-of-sample test dataset:

| Number of Knot | Linear expansion (degree=1) | Cubic expansion (degree=3) |
|---|---|---|
| 1 knot - probs = c(0.5) | 1.9558 | - |
| 2 knots - probs = c(0.33,0.66) | 1.88549 | 1.921419 |
| 3 knots - probs = c(0.25,0.5,0.75) | 1.851085 | 1.839176 |
| 4 knots - probs = c(0.2,0.4,0.6,0.8) | 1.84403 | 1.832675 |
| 5 knots - probs = c(0.16, 0.33, 0.5, 0.66, 0.83) | 1.848455 | 1.833468 |
| 7 knots - probs = c(0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875) | 1.846471 | **1.830979** |

- The cubic polynomial regression model with 7 knots (**model 3**) shows the lowest expected loss on the out-of-sample data (1.830979). Below are the detailed codes to produce **model 3**:

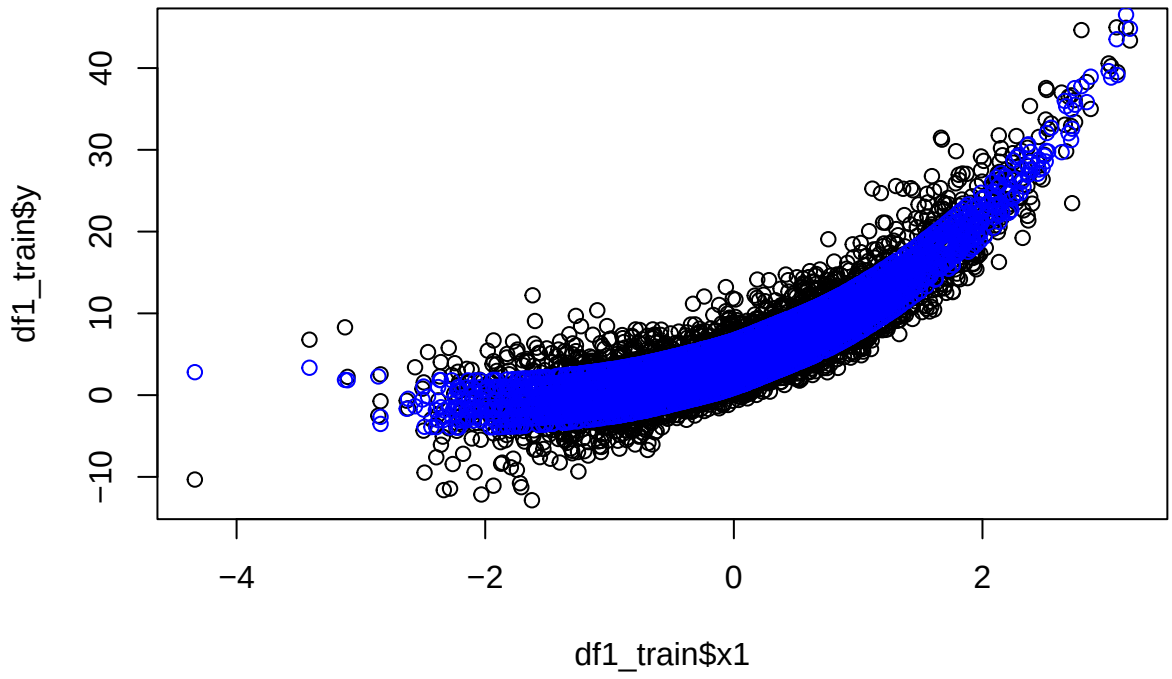
```
# Create knots vector for x1 - 7 knots
knotvec7_df1 = quantile(df1_train$x1, probs=c(0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875))

#create the cubic basis expansion of x1 around 7 knots
```

```
x1_cbe_train_df1 = bs(df1_train$x1, degree=3, knots=knotvec7_df1)

#create the matrix of independent variables
xall_cbe_train_df1 = cbind(ones_train_df1, x1_cbe_train_df1, df1_train$x3, df1_train$x4)

# Regress x-matrix on the linear basis expansion using glmnet
glm1_cbe_df1 = glmnet(xall_cbe_train_df1, df1_train$y, family="gaussian")

# Compute the sample mean absolute loss on the test datset
x1_cbe_test_df1 = bs(df1_test$x1, degree=3, knots=knotvec7_df1)
xall_cbe_test_df1 = cbind(ones_test_df1, x1_cbe_test_df1, df1_test$x3, df1_test$x4)
yhat1_cbe_test_df1 = predict(glm1_cbe_df1, newx=xall_cbe_test_df1, s=0)

loss1_cbe_df1 = abs(df1_test$y - yhat1_cbe_test_df1)
mean(loss1_cbe_df1)
```

```
## [1] 1.830979
```

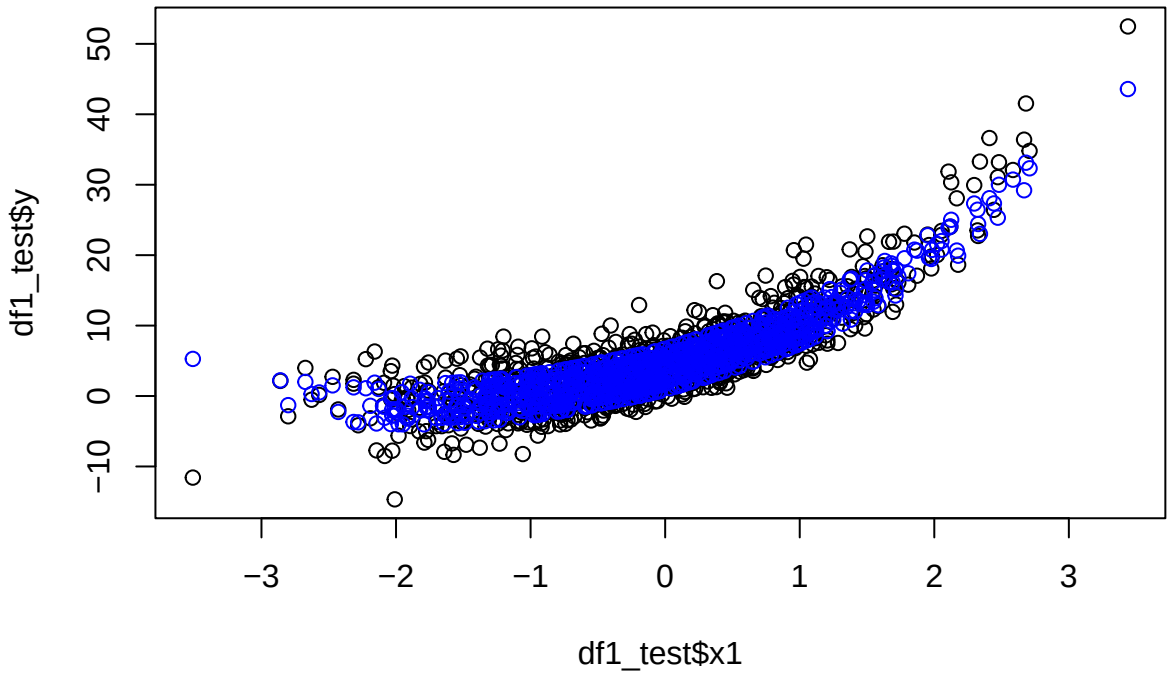
Examining the plot between $x_1$ and $y$ in the training dataset and test dataset (out-of-sample), we can see the cubic spline with 7 knots adds a smooth curve in the dataset, and generally it fits quite well in the out-of-sample dataset (plot below). Therefore, at this stage, **model3** is the best model with lowest value of absolute expected loss.

```
plot(df1_train$x1, df1_train$y)+
points(df1_train$x1, predict(glm1_cbe_df1, newx=xall_cbe_train_df1, s=0), col="blue")
```



```
## integer(0)
```

```
plot(df1_test$x1, df1_test$y)+
points(df1_test$x1, yhat1_cbe_test_df1, col="blue")
```




```
## integer(0)
```

### 3. Cross validation with Lasso and Ridge regression

We now find the optimal model based on cross-validation technique using Lasso and Ridge regression. In which, we will use method *cv.glmnet*() to add additional cross-validation steps to find the optimal value of lambda penalty parameter when we applying Lasso and Ridge regression in the model.

In this case, we will start with the overly complicated model with all 5 explanatory variables. In which, besides applying the cubic basic expansion with 7 knots for $x_1$ as figured out above, we also apply either linear or cubic basic expansion form for $x_2$ and $x_5$ where we acknowledge the increase in the model complexity. Furthermore, the plot between $y$ and $x_2$, $x_5$ does not show any clear trends, and taking into consideration of the curve of dimensionality with a relatively small data set, we will consider the model with the numbers of knots ranging from 2-4 for $x_2$, $x_5$.

Below is the summary of the expected loss of different Lasso regression models with lambda s=0:

| Lasso regression with lambda s=0 | Linear expansion for x2 and x5 | Cubic expansion for x2 and x5 |
|---|---|---|
| 2 knots - probs=c(0.33, 0.66) | 1.828935 | 1.823143 |

| Lasso regression with lambda s=0 | Linear expansion for x2 and x5 | Cubic expansion for x2 and x5 |
|---|---|---|
| 3 knots - probs=c(0.25, 0.5, 0.75) | 1.82881 | **1.821842** |
| 4 knots - probs=c(0.2, 0.4, 0.6, 0.8) | 1.829733 | 1.824149 |

By using the cross validation with Lasso/Ridge regression, the complexity of the model will decrease, particularly, the penalty function will force the parameters towards zero. Based on data-based evidence, models where $x_2$ and $x_5$ are in cubic basic expansion with 3 knots produced the smaller expected loss among others (i.e. 1.821842). For the simplicity of the project, we will only provide the detailed codes of cubic basic expansion for $x_2$ and $x_5$ with 3 knots as below:

- Preparing the x-matrix for training and test dataset where $x_1$ is in cubic basic expansion with 7 knots and $x_2$ and $x_5$ are in cubic basic expansion regression with 3 knots:

```
# Create knots vector for x2 and 5 - 7 knots
knotvec7_df1_x2 = quantile(df1_train$x2, probs=c(0.25, 0.5, 0.75))
knotvec7_df1_x5 = quantile(df1_train$x5, probs=c(0.25, 0.5, 0.75))

#create the matrix of independent variables for training dataset
xall2_be_train_df1 = cbind(ones_train_df1, x1_cbe_train_df1,
                           bs(df1_train$x2, degree=3, knots=knotvec7_df1_x2),
                           df1_train$x3, df1_train$x4,
                           bs(df1_train$x5, degree=3, knots=knotvec7_df1_x5))

#create the matrix of independent variables for out-of-sample dataset
xall2_be_test_df1 = cbind(ones_test_df1, x1_cbe_test_df1,
                           bs(df1_test$x2, degree=3, knots=knotvec7_df1_x2),
                           df1_test$x3, df1_test$x4,
                           bs(df1_test$x5, degree=3, knots=knotvec7_df1_x5))
```

- Building 2 models using cross-validation $cv.glmnet()$, where the first one is built with Lasso penalty function ($alpha = 1$) as a default value and the latter is built with Ridge penalty function ($alpha = 0$).

```
# Creating model under lasso/ridge regression using cross-validation cv.glmnet
glm_lasso_df1 = cv.glmnet(xall2_be_train_df1, df1_train$y, family="gaussian")
glm_ridge_df1 = cv.glmnet(xall2_be_train_df1, df1_train$y, family="gaussian", alpha=0)
```

- Computing the prediction on the test dataset for each of the model above. We also consider different value of lambda where:
  - $s = 0$: no penalty
  - $s = lambda.min$: generating predictions with cross-validated optimal value for lambda penalty parameter
  - $s = lambda.1se$: generating predictions with alternate lambda values that is larger than lambda.min but still with one standard error

```
# Computing the prediction on the out-of-sample dataset for Lasso regression
yhat_test_lasso1_df1 = predict(glm_lasso_df1, newx=xall2_be_test_df1, s=0)
loss1_lasso_df1 = abs(df1_test$y - yhat_test_lasso1_df1)
mean(loss1_lasso_df1)
```

```
## [1] 1.821842
```

```
yhat_test_lasso2_df1 = predict(glm_lasso_df1, newx=xall2_be_test_df1, s="lambda.min")
loss2_lasso_df1 = abs(df1_test$y - yhat_test_lasso2_df1)
mean(loss2_lasso_df1)
```

```
## [1] 1.830471
```

```
yhat_test_lasso3_df1 = predict(glm_lasso_df1, newx=xall2_be_test_df1, s="lambda.1se")
loss3_lasso_df1 = abs(df1_test$y - yhat_test_lasso3_df1)
mean(loss3_lasso_df1)
```

```
## [1] 1.85647
```

```
#for Ridge regression
yhat_test_ridge1_df1 = predict(glm_ridge_df1, newx=xall2_be_test_df1, s=0)
loss1_ridge_df1 = abs(df1_test$y - yhat_test_ridge1_df1)
mean(loss1_ridge_df1)
```

```
## [1] 1.83412
```

```
yhat_test_ridge2_df1 = predict(glm_ridge_df1, newx=xall2_be_test_df1, s="lambda.min")
loss2_ridge_df1 = abs(df1_test$y - yhat_test_ridge2_df1)
mean(loss2_ridge_df1)
```

```
## [1] 1.83412
```

```
yhat_test_ridge3_df1 = predict(glm_ridge_df1, newx=xall2_be_test_df1, s="lambda.1se")
loss3_ridge_df1 = abs(df1_test$y - yhat_test_ridge3_df1)
mean(loss3_ridge_df1)
```

```
## [1] 1.860937
```

**Comment**:

- Among models with different value of lambda, model with Lasso penalty function where value of lambda s = 0 (so called **model 4**) produces a lower absolute expected loss = **1.821842**.
- Comparing to model 3 using step-wise method where the absolute expected loss = 1.830979, model 4 also yields a lower absolute expected loss.
- Therefore, we conclude that **model 4** is the optimal model for **df1**, minimizing the magnitude of the prediction errors and thus yield the most accurate predictions of the target variable $y$.

### 1.5. Prediction

Now, using the optimal **model 4**, we can compute the prediction on the unseen data. We are provided with 5 observations in the dataset **predict_df1** as imported below:

```
predict_df1 <- read_csv("df1_quant_y_predict.csv")
predict_df1
```

```
## # A tibble: 5 x 5
##        x1    x2    x3    x4     x5
##     <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1  0.654   4.07     0     1  0.763
## 2 -1.15    9.69     1     0  1.41
## 3 -0.983   4.67     3     0  1.18
## 4 -0.472   3.10     1     1 -1.00
## 5 -0.0342  5.68     1     1  0.698
```

Before performing the prediction, we would want to manipulate explanatory variables of the predicted dataset as follow:

```
# create vector of ones for the predict dataset
ones_predict_df1 = rep(1, nrow(predict_df1))
# create the explanatory matrix
xall_new_df1 = cbind(ones_predict_df1,
                     bs(predict_df1$x1, degree=3, knots=knotvec7_df1),
                     bs(predict_df1$x2, degree=3, knots=knotvec7_df1_x2),
                     predict_df1$x3, predict_df1$x4,
                     bs(predict_df1$x5, degree=3, knots=knotvec7_df1_x5))
```

Now, we are able to produce the prediction using the optimal **model4**, with the result for index 1 to 5 as below:

```
yhat_new_lasso_df1 = predict(glm_lasso_df1, newx=xall_new_df1, s=0)
yhat_new_lasso_df1
```

```
##             s1
## [1,] 19.062313
## [2,]  2.506744
## [3,] -2.929043
## [4,]  3.446199
## [5,]  5.235450
```

## II. Dataset with categorical target variable

### 2.1. Import data:

The data with categorical dependent variable is imported as **df2**. There are 4,800 observations in the dataset, in which $x_1, x_2$ and $x_5$ are continuous variables, where $x_3$ and $x_4$ are categorical variables.

Furthermore, as we are dealing with categorical dependent variable $y$, we would want to convert $y$ to factor. A quick look to the dataset is as below:

```
df2 = read.csv("df2_cat_y.csv")
df2 <- df2 %>%
  mutate(y = factor(y))
head(df2)
```

```
##       y         x1         x2 x3 x4        x5
## 1   red  1.1279977 0.33852851  0  0 1.8491712
## 2 green -1.1582409 0.65338157  1  1 0.9543027
## 3  blue -1.8513715 1.61836922  1  1 1.3315813
## 4   red  0.4591725 0.08311089  1  1 1.6847083
## 5   red -0.6857287 0.52022258  0  0 0.8207516
## 6   red  0.6435366 0.40475201  4  0 0.8450380
```

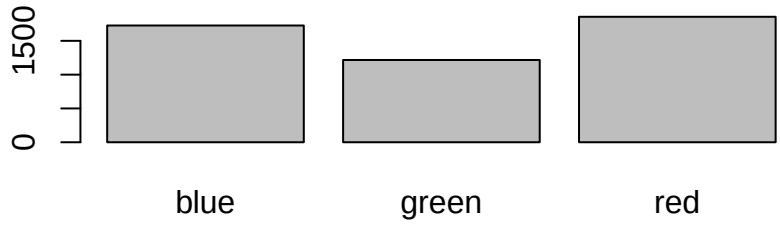
```
nrow(df2) #number of observations
```

```
## [1] 4800
```

### 2.2. Graphical examination:

Graphically examining the plot of target variable $y$ and explanatory variables:

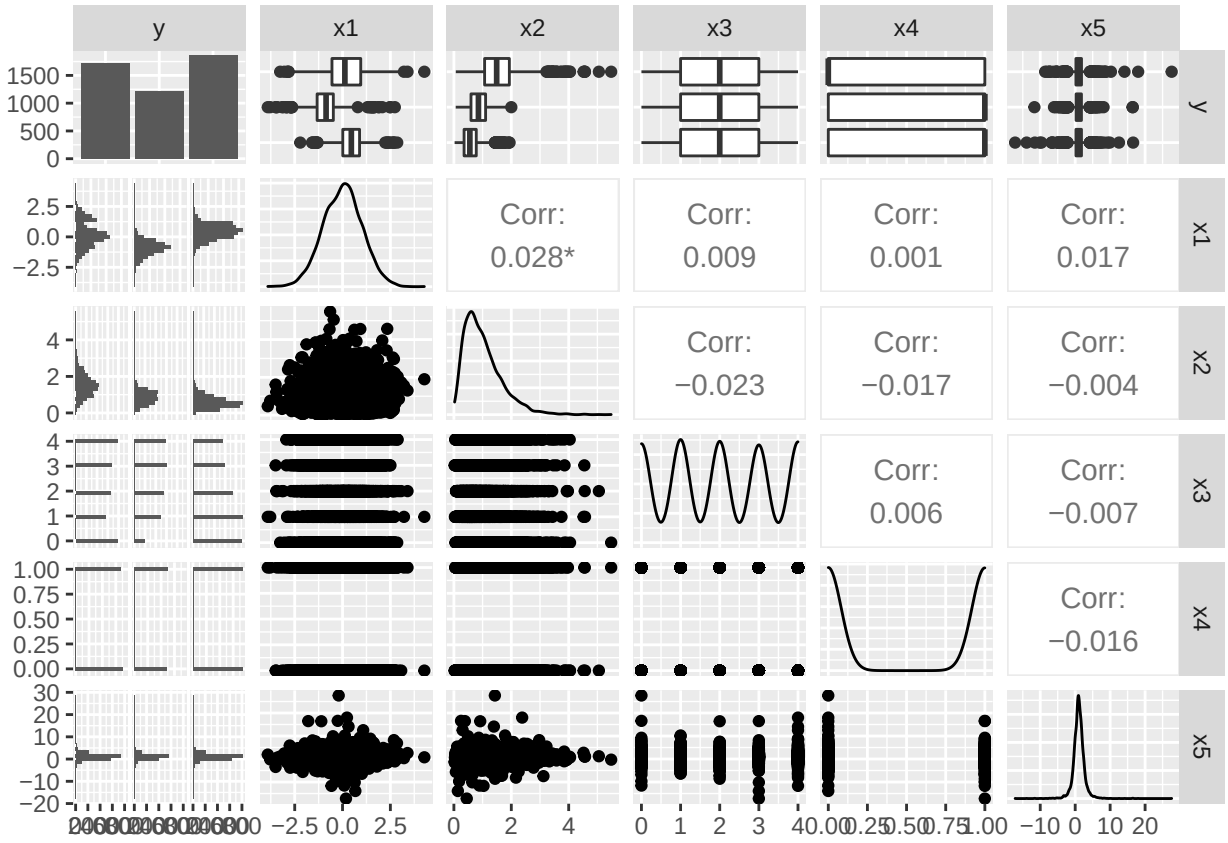
```
plot(df2$y)
```




- The class in target variable is quite balance.
- We can also look at the ggpairs plot of df2 (using package GGally), where we can see some trend of $y$ with $x_1$ and $x_2$, while $x_5$ shows no trend with different classes of target variable $y$.

```
ggpairs(df2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




### 2.3. Data manipulation

We will split the data into training and test data sets according to the proportion of 80/20. Then, we create the column of ones having the same length as training and test data sets.

```
train_df2 = df2[1:3840, ] #80% train and 20% test
test_df2 = df2[3841:nrow(df2), ]

ones_train_df2 = rep(1, nrow(train_df2))
ones_test_df2 = rep(1, nrow(test_df2))
```

### 2.4. Model Building

Regarding categorical dependent variable, we would want to try the following strategy:

- build a simple model **without** any basic expansion with cross-validation using Lasso and Ridge Regression
- build a complex model **with** basic expansion and cross-validation using Lasso and Ridge Regression.
- build model using K-Nearest Neighbors algorithm.
- then, we compare the proportion of correct and incorrect classifications and choose the model that result the highest accuracy.

### 1. Model without basic expansion

To build the model using *glmnet*(), we firstly create the matrix of explanatory variables:

```
xall_train_df2 = cbind(ones_train_df2, train_df2$x1, train_df2$x2,
                       train_df2$x3, train_df2$x4, train_df2$x5)
```

Then, we now use a logistic regression to regress $y$ on explanatory variables using the training data:

```
# Logit binomial without basis expansion
glm_df2_lasso1 = glmnet(xall_train_df2, train_df2$y, family="multinomial") #Lasso regression
glm_df2_ridge1 = glmnet(xall_train_df2, train_df2$y, family="multinomial", alpha=0) #Ridge regression
```

Now, we compute the model on the out-of-sample dataset and get the proportion of correct classification:

- Creating the matrix of explanatory variable of test dataset:

```
# Get the out-of-sample proportion of correct classifications
xall_test_df2 = cbind(ones_test_df2, test_df2$x1, test_df2$x2,
                      test_df2$x3, test_df2$x4, test_df2$x5)
```

- Use the 2 models above *glm_df2_lasso1* and *glm_df2_ridge1* to make predictions on the test dataset (s=0 ensure we will not apply lambda penalty prediction).

```
yhat_test_df2_lasso = predict(glm_df2_lasso1, newx=xall_test_df2, type="class", s=0)
yhat_test_df2_ridge = predict(glm_df2_ridge1, newx=xall_test_df2, type="class", s=0)

outcome_probabilities_lasso = predict(glm_df2_lasso1, newx=xall_test_df2, s=0, type="response")
outcome_probabilities_ridge = predict(glm_df2_ridge1, newx=xall_test_df2, s=0, type="response")
```

- Checking the proportion of correct and incorrect classifications of the 2 models:

```
#Proportion of correct classification:
sum(test_df2$y == yhat_test_df2_lasso) / nrow(test_df2)
```

```
## [1] 0.8083333
```

```
sum(test_df2$y == yhat_test_df2_ridge) / nrow(test_df2)
```

```
## [1] 0.809375
```

```
# indicator of misclassifications:
incorrect_classification_lasso1 = sum(test_df2$y != yhat_test_df2_lasso)
correct_classification_lasso1 = sum(test_df2$y == yhat_test_df2_lasso)
cbind(correct_classification_lasso1, incorrect_classification_lasso1)
```

```
##      correct_classification_lasso1 incorrect_classification_lasso1
## [1,]                           776                             184
```

```
incorrect_classification_ridge1 = sum(test_df2$y != yhat_test_df2_ridge)
correct_classification_ridge1 = sum(test_df2$y == yhat_test_df2_ridge)
cbind(correct_classification_ridge1, incorrect_classification_ridge1)
```

```
##      correct_classification_ridge1 incorrect_classification_ridge1
## [1,]                           777                             183
```

From the above, Lasso regression model has smaller number of proportion of correct classification (80.83%) comparing to the Ridge regression model (80.93%) . As a result, Lasso regression model has higher number of incorrect observations (184) while Ridge regression model has 183 incorrect observations, meaning that Ridge regression (so called **model1**) performs slightly better than Lasso regression in the model without basic expansion for continuous variables.

Next, we should add basic expansion for the continuous variables in the model to see if it increases the model prediction accuracy.

**2. Model with basic expansion for x1 and x2**

- As explained in the data exploration, $x_1$ and $x_2$ might have relationships with $y$ while $x_5$ shows no relationship. Therefore, to keep the model simple, we will try the linear and cubic basic expansion for $x_1$ and $x_2$ with different knots (2-5).
- We can build a simple model with linear basic expansion and 2 knots as below:

```
# Get a linear basis expansion of x1 and x2 around 2 knots
knotvec_x1_df2 = quantile(train_df2$x1, probs=c(0.33,0.66))
knotvec_x2_df2 = quantile(train_df2$x2, probs=c(0.33,0.66))
xmat_x1_train_df2 = bs(train_df2$x1, degree=1, knots=knotvec_x1_df2)
xmat_x2_train_df2 = bs(train_df2$x2, degree=1, knots=knotvec_x2_df2)

# Create the explanatory variables matrix
xmat_train_df2_all = cbind(ones_train_df2,xmat_x1_train_df2 , xmat_x2_train_df2,
                           train_df2$x3, train_df2$x4,train_df2$x5)

# Regress x1 and x2 on the linear basis expansion using glmnet
df2_lbe_lasso = glmnet(xmat_train_df2_all, train_df2$y, family="multinomial")#Lasso
df2_lbe_ridge = glmnet(xmat_train_df2_all, train_df2$y, family="multinomial", alpha=0)# Ridge

# Compute the sample mean absolute loss on the test set
xmat_x1_test_df2 = bs(test_df2$x1, degree=1, knots=knotvec_x1_df2)
xmat_x2_test_df2 = bs(test_df2$x2, degree=1, knots=knotvec_x2_df2)

# Create the explanatory variables matrix for test set
xmat_test_df2_all = cbind(ones_test_df2,xmat_x1_test_df2 , xmat_x2_test_df2,
                           test_df2$x3, test_df2$x4,test_df2$x5)
# Predict
yhat_cbe_df2_cat_lasso = predict(df2_lbe_lasso, newx=xmat_test_df2_all, s=0, type="class")
yhat_cbe_df2_cat_ridge = predict(df2_lbe_ridge, newx=xmat_test_df2_all, s=0, type="class")
```

```
# Number of correct classifications
sum(test_df2$y == yhat_cbe_df2_cat_lasso) / nrow(test_df2)
```

```
## [1] 0.8302083
```

```
sum(test_df2$y == yhat_cbe_df2_cat_ridge) / nrow(test_df2)
```

```
## [1] 0.8104167
```

```
# indicator of misclassifications:
incorrect_classification_lasso2 = sum(test_df2$y != yhat_cbe_df2_cat_lasso)
correct_classification_lasso2 = sum(test_df2$y == yhat_cbe_df2_cat_lasso)
cbind(correct_classification_lasso2, incorrect_classification_lasso2)
```

```
##      correct_classification_lasso2 incorrect_classification_lasso2
## [1,]                           797                             163
```

```
incorrect_classification_ridge2 = sum(test_df2$y != yhat_cbe_df2_cat_ridge)
correct_classification_ridge2 = sum(test_df2$y == yhat_cbe_df2_cat_ridge)
cbind(correct_classification_ridge2, incorrect_classification_ridge2)
```

```
##      correct_classification_ridge2 incorrect_classification_ridge2
## [1,]                           778                             182
```

- Between these 2 models, Lasso regression model with linear basic expansion around 2 knots for $x_1$ and $x_2$ (so called **model2**) performs better than the one with Ridge regression.
- Comparing to model1, **model2** performs better where the proportion of correct classification increase from 0.809375 to 0.8302 and number of incorrect classifications reduce from 183 to 163 observations.
- Similarly, we can compute the proportion of correct classification of Lasso and Ridge regression with Linear/Cubic basic expansion across 2-5 knots for $x_1$ and $x_2$. Below is the summary table:

| Number of knot | Linear expansion (degree=1) | | Cubic expansion (degree=3) | |
|---|---|---|---|---|
| | **Lasso** | • *R idge** | **Lasso** | **Ridge** |
| 2 knots | 0.8302083 | 0 .8104167 | 0.828125 | 0.8270833 |
| 3 knots | 0.840625 | 0.821875 | 0.83125 | 0.8385417 |
| 4 knots | 0.85625 | 0 .8302083 | 0.8541667 | 0.8489583 |
| 5 knots | **0.8614583** | 0.85 | 0.8510417 | 0.8510417 |

- Lasso regression model with linear basic expansion with 5 knots (so called **model3)** results the highest proportion of correct classification (**0.8614583**) among alternatives. Below is the detailed codes for **model3**:

```
# Get a linear basis expansion of x1 and x2 around 5 knots
knotvec_x1_df25 = quantile(train_df2$x1, probs=c(0.16, 0.33, 0.5, 0.66, 0.83))
knotvec_x2_df25 = quantile(train_df2$x2, probs=c(0.16, 0.33, 0.5, 0.66, 0.83))

xmat_x1_train_df25 = bs(train_df2$x1, degree=1, knots=knotvec_x1_df25)
xmat_x2_train_df25 = bs(train_df2$x2, degree=1, knots=knotvec_x2_df25)

#Create the explanatory variables matrix
xmat_train_df2_all5 = cbind(ones_train_df2,xmat_x1_train_df25 , xmat_x2_train_df25,
```

```
                          train_df2$x3, train_df2$x4,train_df2$x5)

# Regress x1 and x2 on the linear basis expansion using glmnet
df2_cbe_lasso5 = glmnet(xmat_train_df2_all5, train_df2$y, family="multinomial")
df2_cbe_ridge5 = glmnet(xmat_train_df2_all5, train_df2$y, family="multinomial", alpha=0)

# Compute the sample mean absolute loss on the test set
xmat_x1_test_df25 = bs(test_df2$x1, degree=1, knots=knotvec_x1_df25)
xmat_x2_test_df25 = bs(test_df2$x2, degree=1, knots=knotvec_x2_df25)

#Create the explanatory variables matrix for test set
xmat_test_df2_all5 = cbind(ones_test_df2,xmat_x1_test_df25 , xmat_x2_test_df25,
                          test_df2$x3, test_df2$x4,test_df2$x5)
# Predict
yhat_cbe_df2_cat_lasso5 = predict(df2_cbe_lasso5, newx=xmat_test_df2_all5, s=0, type="class")
yhat_cbe_df2_cat_ridge5 = predict(df2_cbe_ridge5, newx=xmat_test_df2_all5, s=0, type="class")

# Number of correct classifications
sum(test_df2$y == yhat_cbe_df2_cat_lasso5) / nrow(test_df2)
```

```
## [1] 0.8614583
```

```
sum(test_df2$y == yhat_cbe_df2_cat_ridge5) / nrow(test_df2)
```

```
## [1] 0.85
```

```
# indicator of misclassifications:
incorrect_classification_lasso5 = sum(test_df2$y != yhat_cbe_df2_cat_lasso5)
correct_classification_lasso5 = sum(test_df2$y == yhat_cbe_df2_cat_lasso5)
cbind(correct_classification_lasso5, incorrect_classification_lasso5)
```

```
##      correct_classification_lasso5 incorrect_classification_lasso5
## [1,]                           827                             133
```

```
incorrect_classification_ridge5 = sum(test_df2$y != yhat_cbe_df2_cat_ridge5)
correct_classification_ridge5 = sum(test_df2$y == yhat_cbe_df2_cat_ridge5)
cbind(correct_classification_ridge5, incorrect_classification_ridge5)
```

```
##      correct_classification_ridge5 incorrect_classification_ridge5
## [1,]                           816                             144
```

- Having 86.15% correct classification, the incorrect classifications under **model3** significantly have been reduces to 133 observations, indicating a good model for df2.
- For prudent, we would want to build models with linear expansion with 5 knots for continuous variable $x_5$ as similar as $x_1$ and $x_2$ and compare the result with model3.

```
# Get a linear basis expansion of x5 around 5 knots
knotvec_x5_df25 = quantile(train_df2$x5, probs=c(0.16, 0.33, 0.5, 0.66, 0.83))
xmat_x5_train_df25 = bs(train_df2$x5, degree=1, knots=knotvec_x5_df25)
```

```
#Create the explanatory variables matrix
xmat_train_df2_all5b = cbind(ones_train_df2, xmat_x1_train_df25, xmat_x2_train_df25,
                           train_df2$x3, train_df2$x4,
                           xmat_x5_train_df25)

# Regress x1 and x2 on the linear basis expansion using glmnet
df2_cbe_lasso5b = glmnet(xmat_train_df2_all5b, train_df2$y, family="multinomial")
df2_cbe_ridge5b = glmnet(xmat_train_df2_all5b, train_df2$y, family="multinomial", alpha=0)

# Compute the sample mean absolute loss on the test set
xmat_x5_test_df25 = bs(test_df2$x5, degree=1, knots=knotvec_x5_df25)

#Create the explanatory variables matrix for test set
xmat_test_df2_all5b = cbind(ones_test_df2,xmat_x1_test_df25 , xmat_x2_test_df25,
                           test_df2$x3, test_df2$x4,
                           xmat_x5_test_df25)
# Predict
yhat_cbe_df2_cat_lasso5b = predict(df2_cbe_lasso5b, newx=xmat_test_df2_all5b, s=0, type="class")
yhat_cbe_df2_cat_ridge5b = predict(df2_cbe_ridge5b, newx=xmat_test_df2_all5b, s=0, type="class")

# Number of correct classifications
sum(test_df2$y == yhat_cbe_df2_cat_lasso5b) / nrow(test_df2)
```

```
## [1] 0.8666667
```

```
sum(test_df2$y == yhat_cbe_df2_cat_ridge5b) / nrow(test_df2)
```

```
## [1] 0.8510417
```

```
# indicator of misclassifications:
incorrect_classification_lasso5b = sum(test_df2$y != yhat_cbe_df2_cat_lasso5b)
correct_classification_lasso5b = sum(test_df2$y == yhat_cbe_df2_cat_lasso5b)
cbind(correct_classification_lasso5b, incorrect_classification_lasso5b)
```

```
##      correct_classification_lasso5b incorrect_classification_lasso5b
## [1,]                            832                              128
```

```
incorrect_classification_ridge5b = sum(test_df2$y != yhat_cbe_df2_cat_ridge5b)
correct_classification_ridge5b = sum(test_df2$y == yhat_cbe_df2_cat_ridge5b)
cbind(correct_classification_ridge5b, incorrect_classification_ridge5b)
```

```
##      correct_classification_ridge5b incorrect_classification_ridge5b
## [1,]                            817                              143
```

Model **df2__cbe__lasso5b (model4)** performs better than model3 with 86.67% correctly predict out-of-sample data, reducing the number of incorrect classification to 128 observations in comparison to 133 observations from model3. Nevertheless, model4 is more complex than model3 with higher dimensional. For model with high complexity, it will increase the bias, leading to overfitting to unseen data. Whereas, if the model is too simple, it could not capture the non-linearity of the variables, the variance will increase leading to an underfitting model. Hence, bias-variance trade-off should be considered when select the optimal model.

### 3. KNN algorithm

Nearest Neighbor algorithm is a classification method using for classification and regression, where an observation is classified and assigned to the most common class among its k-nearest neighbors (i.e. k to denote the size of the nearest neighbors).

In order to implement K-NN algorithm, we need to re-scale all input data into interval [0,1]. We can define a function to re-scale as below:

```
# Define a function for re-scaling
scale_func = function(x) {(x - min(x)) / (max(x) - min(x))}

# Re-scale all explanatory variables data to the [0,1] interval

df2$x1 = scale_func(df2$x1)
df2$x2 = scale_func(df2$x2)
df2$x3 = scale_func(df2$x3)
df2$x4 = scale_func(df2$x4)
df2$x5 = scale_func(df2$x5)
head(df2)
```

```
##       y        x1         x2   x3 x4        x5
## 1   red 0.6152783 0.05740488 0.00  0 0.4250229
## 2 green 0.3359427 0.11512315 0.25  1 0.4050459
## 3  blue 0.2512550 0.29202289 0.25  1 0.4134682
## 4   red 0.5335604 0.01058220 0.25  1 0.4213514
## 5   red 0.3936748 0.09071270 0.00  0 0.4020645
## 6   red 0.5560862 0.06954485 1.00  0 0.4026067
```

Next, we split the data set again into training and test data sets

```
# Split the dataset up into training and test set
df2_train_knn = df2[1:3840, ]
df2_test_knn = df2[3841:nrow(df2), ]
```

```
# Prepare the input matrix for training set
xall_train = cbind(df2_train_knn$x1, df2_train_knn$x2,
                   df2_train_knn$x3,df2_train_knn$x4,df2_train_knn$x5)

# Prepare the input matrix for test set
xall_test = cbind(df2_test_knn$x1, df2_test_knn$x2,
                  df2_test_knn$x3,df2_test_knn$x4,df2_test_knn$x5)

# Factorize the target variable in training set.
y_train = factor(df2_train_knn$y)
```

Using method knn() in package class, we can generate the prediction on test dataset.

The most important value in the model is the number of k. If the k-value is too small, it will increase the chance of over fitting. Therefore, we can compute the number of k ranging from 8-15 for this dataset.

```
library(class)
yhat_test_8 = knn(xall_train, xall_test, y_train, k=8)
yhat_test_10 = knn(xall_train, xall_test, y_train, k=10)
```

```
yhat_test_12 = knn(xall_train, xall_test, y_train, k=12)
yhat_test_15 = knn(xall_train, xall_test, y_train, k=15)

sum(df2_test_knn$y == yhat_test_8) / nrow(df2_test_knn)
```

```
## [1] 0.9322917
```

```
sum(df2_test_knn$y == yhat_test_10) / nrow(df2_test_knn)
```

```
## [1] 0.9354167
```

```
sum(df2_test_knn$y == yhat_test_12) / nrow(df2_test_knn)
```

```
## [1] 0.9333333
```

```
sum(df2_test_knn$y == yhat_test_15) / nrow(df2_test_knn)
```

```
## [1] 0.928125
```

Model with number of k = 10 is the optimal model (**model5**) with the highest proportion of correct classification on the out-of-sample data. The percentage of correction classification of model5 is higher than **model4** (i.e. 86.66%) using Lasso regression with basic expansion.

**Conclusion**: The data analyst should consider the business acumen and industrial knowldege to choose the optimal ML model for each type of dataset. For example, in medical or financial industry, the model will focus on reducing the number of incorrect classification *(i.e. wrongly predict a normal person that has cancer is still better than incorrectly predicting a person (who has cancer) that does not have the disease),* thus, model 5 will be of interest. However, in marketing domain where the generality is considered, model 4 might perform better.

For the purpose of producing the model that minimize the expected loss to increase the prediction accuracy, we will opt for **model 5** and use it to predict the unseen data.

### 2.5. Prediction

Using the optimal **model 5**, we compute the prediction on the unseen data as imported below.

```
df2_predict <- read_csv("df2_cat_y_predict.csv")
df2_predict
```

```
## # A tibble: 5 x 5
##       x1    x2    x3    x4     x5
##    <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 -0.599 1.23      4     1 -0.590
## 2  0.834 0.788     4     0  0.515
## 3 -1.41  0.866     0     1  1.85
## 4  0.337 2.20      1     0  7.92
## 5 -0.261 0.914     0     1  2.15
```

We prepare the matrix for the explanatory variables and apply the KNN algorithm with k-10 as below:

```r
# Scale input using function scale_func defined above
df2_predict$x1 = scale_func(df2_predict$x1)
df2_predict$x2 = scale_func(df2_predict$x2)
df2_predict$x3 = scale_func(df2_predict$x3)
df2_predict$x4 = scale_func(df2_predict$x4)
df2_predict$x5 = scale_func(df2_predict$x5)
df2_predict
```

```
## # A tibble: 5 x 5
##      x1     x2    x3    x4    x5
##   <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 0.363 0.312   1         1 0
## 2 1     0       1         0 0.130
## 3 0     0.0554  0         1 0.287
## 4 0.779 1       0.25      0 1
## 5 0.513 0.0896  0         1 0.322
```

```r
# Create the matrix of input variable
xall_predict_df2 = cbind(df2_predict$x1, df2_predict$x2,
                         df2_predict$x3,df2_predict$x4,df2_predict$x5)
# Predict
yhat_valid_opt = knn(xall_train, xall_predict_df2, factor(df2_train_knn$y), k=10)
yhat_valid_opt
```

```
## [1] blue blue blue blue red
## Levels: blue green red
```

Above is the prediction result of the 5 unseen observations with the order as below:

| Index of x | Predicted value of y |
|---|---|
| 1 | blue |
| 2 | blue |
| 3 | blue |
| 4 | blue |
| 5 | red |