Loading libraries required and reading the data into R

```
library(knitr)
library(ggplot2)
library(plyr)
library(dplyr)
library(corrplot)
library(caret)
library(gridExtra)
library(scales)
library(Rmisc)
library(ggrepel)
library(psych)
library(xgboost)
library(glmnet)
library(gbm)

train <- read.csv("train.csv",stringsAsFactors = F)
test <- read.csv("test.csv",stringsAsFactors = F)
```

Separate corresponding Sale_Price and PIDs from raw training and test data, and then append processed training and test data into a single matrix, to go though feature engineering together. The benefit of binding them together is to circumvent mismatch betweeen categorical variable levels of training and test data during preprocessing, which will be elaborated later.

```
trainprice=train$Sale_Price
train$Sale_Price=NULL
alldata=rbind(train,test)
alldata=alldata[,-c(1)]
trainpid=train[,1]
testpid=test[,1]
```

Feature Engineering

I herein deal with possible NA values by imputing them with the most frequent patterns of the corresponding variables,regardless of NAs, i.e. mode numbers. Thus, a function to determine mode numbers is hereby defined for later use:

```
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

There are basically 3 types of variables in the raw data: numericals, ordinal characters and categorical character variables, each of which deserves respective preprocessing. For sake of compactness of the code, all feature engineering steps are encapsulated into the customized function of processing:

```r
processing=function(all){
# First, numerical variables are sought out:
  numericVars <- which(sapply(all, is.numeric)) #index vector numeric variables
  numericVarNames <- names(numericVars) #saving names vector for use later on
  all_numVar <- all[, numericVars]
# Sequentially, implement label encoding/factorizing the remaining character variables, based upon the ordinality.
  Charcol <- names(all[,sapply(all, is.character)])
# I targeted ordinal variables according to the variable description, and label-encoded them regarding their levels.
  Ordinalnames=c(
    'Overall_Qual','Overall_Cond','Lot_Shape','Exter_Qual','Exter_Cond','Bsmt_Qual','Bsmt_Cond',
    'Bsmt_Exposure','BsmtFin_Type_1','BsmtFin_Type_2','Heating_QC','Electrical','Kitchen_Qual',
    'Functional','Fireplace_Qu','Garage_Qual','Garage_Cond','Paved_Drive','Fence'
  )


  all$Overall_Qual<-as.integer(revalue(all$Overall_Qual, c("Very_Excellent"=10, "Excellent"=9,
                                                           "Very_Good"=8,"Good"=7,"Above_Average"=6,"Average"=5, "Below_Average"=4,
                                                           "Fair"=3, "Poor"=2   ,"Very_Poor"=1             )))


  all$Overall_Cond<-as.integer(revalue(all$Overall_Cond, c("Excellent"=9,
                                                           "Very_Good"=8,"Good"=7,"Above_Average"=6,"Average"=5, "Below_Average"=4,
                                                           "Fair"=3, "Poor"=2   ,"Very_Poor"=1             )))
  all$Lot_Shape<-as.integer(revalue(all$Lot_Shape, c("Irregular"=0,
                                                     "Moderately_Irregular"=1,"Slightly_Irregular"=2,"Regular"=3)))
  all$Exter_Qual<-as.integer(revalue(all$Exter_Qual, c("Fair"=0,"Typical"=1,"Good"=2,"Excellent"=3)))
  all$Exter_Cond<-as.integer(revalue(all$Exter_Cond, c("Poor"=0,"Fair"=1,"Typical"=2,"Good"=3,"Excellent"=4)))
  all$Bsmt_Qual<-as.integer(revalue(all$Bsmt_Qual, c("No_Basement"=0,"Poor"=1,"Fair"=2,"Typical"=3,"Good"=4,"Excellent"=5)))
  all$Bsmt_Cond<-as.integer(revalue(all$Bsmt_Cond, c("No_Basement"=0,"Poor"=1,"Fair"=2,"Typical"=3,"Good"=4,"Excellent"=5)))
  all$Bsmt_Exposure<-as.integer(revalue(all$Bsmt_Exposure, c("No_Basement"=0,"No"=1,"Mn"=2,"Av"=3,"Gd"=4)))
  all$BsmtFin_Type_1<-as.integer(revalue(all$BsmtFin_Type_1, c("No_Basement"=0,"Unf"=1,"LwQ"=2,"Rec"=3,"BLQ"=4,'ALQ'=5,'GLQ'=6)))
  all$BsmtFin_Type_2<-as.integer(revalue(all$BsmtFin_Type_2, c("No_Basement"=0,"Unf"=1,"LwQ"=2,"Rec"=3,"BLQ"=4,'ALQ'=5,'GLQ'=6)))
  all$Heating_QC<-as.integer(revalue(all$Heating_QC, c("Poor"=0,"Fair"=1,"Typical"=2,"Good"=3,"Excellent"=4)))
  all$Electrical<-as.integer(revalue(all$Electrical, c("Mix"=0,"FuseP"=1,"FuseF"=2,"FuseA"=3,"SBrkr"=4,"Unknown"=2)))
  all$Kitchen_Qual<-as.integer(revalue(all$Kitchen_Qual, c("Poor"=0,"Fair"=1,"Typical"=2,"Good"=3,"Excellent"=4)))
  all$Functional<-as.integer(revalue(all$Functional, c("Sal"=0,"Sev"=1,"Maj2"=2,"Maj1"=3,"Mod"=4,"Min2"=5,"Min1"=6,"Typ"=7)))
  all$Fireplace_Qu<-as.integer(revalue(all$Fireplace_Qu, c("No_Fireplace"=0,"Poor"=1,"Fair"=2,"Typical"=3,"Good"=4,"Excellent"=5)))
  all$Garage_Finish<-as.integer(revalue(all$Garage_Finish, c("No_Garage"=0,"Unf"=1,"RFn"=2,"Fin"=3)))
```

```r
  all$Garage_Qual<-as.integer(revalue(all$Garage_Qual, c("No_Garage"=0,"Poor"=1,"Fair"=2,"Typical"=3,"Good"=4,"Excellent"=5)))

  all$Garage_Cond<-as.integer(revalue(all$Garage_Cond, c("No_Garage"=0,"Poor"=1,"Fair"=2,"Typical"=3,"Good"=4,"Excellent"=5)))

  all$Paved_Drive<-as.integer(revalue(all$Paved_Drive, c("Dirt_Gravel"=0,"Partial_Pavement"=1,"Paved"=2)))

  all$Fence<-as.integer(revalue(all$Fence, c("No_Fence"=0,"Minimum_Wood_Wire"=1,"Good_Wood"=2,"Minimum_Privacy"=3,"Good_Privacy"=4)))


# Besides, for categorical variables, factorizing and one-hot encoding was implemented as below:

# 1.factorization:

  factornames=c(

    'MS_SubClass','MS_Zoning','Alley','Street', 'Land_Contour', 'Lot_Config', 'Neighborhood','Condition_1',

    'Condition_2', 'Bldg_Type', 'House_Style','Roof_Style', 'Roof_Matl','Exterior_1st', 'Exterior_2nd',

    'Mas_Vnr_Type', 'Foundation', 'Heating','Central_Air','Garage_Type','Misc_Feature', 'Sale_Type',

    'Sale_Condition', 'Mo_Sold','Year_Sold')


  all$MS_SubClass=as.factor(all$MS_SubClass)

  all$MS_Zoning=as.factor(all$MS_Zoning)

  all$Alley=as.factor(all$Alley)

  all$Street=as.factor(all$Street)

  all$Land_Contour=as.factor(all$Land_Contour)

  all$Lot_Config=as.factor(all$Lot_Config)

  all$Neighborhood=as.factor(all$Neighborhood)

  all$Condition_1=as.factor(all$Condition_1)

  all$Condition_2=as.factor(all$Condition_2)

  all$Bldg_Type=as.factor(all$Bldg_Type)

  all$House_Style=as.factor(all$House_Style)

  all$Roof_Style=as.factor(all$Roof_Style)

  all$Roof_Matl=as.factor(all$Roof_Matl)

  all$Exterior_1st=as.factor(all$Exterior_1st)

  all$Exterior_2nd=as.factor(all$Exterior_2nd)

  all$Mas_Vnr_Type=as.factor(all$Mas_Vnr_Type)

  all$Foundation=as.factor(all$Foundation)

  all$Heating=as.factor(all$Heating)

  all$Central_Air=as.factor(all$Central_Air)

  all$Garage_Type=as.factor(all$Garage_Type)

  all$Misc_Feature=as.factor(all$Misc_Feature)

  all$Sale_Type=as.factor(all$Sale_Type)

  all$Sale_Condition=as.factor(all$Sale_Condition)

  all$Mo_Sold=as.factor(as.factor(all$Mo_Sold))

  all$Year_Sold=as.factor(as.factor(all$Year_Sold))


# 2.One-Hot-Encoding

  DFdummies <- as.data.frame(model.matrix(~.-1, DFfactors))
```

```r
# By now, all predictor variables are converted into meaningful numericals. We combine them into a new predictor
 matrix to be taken into the prediction model later on.
  DFfactors <- all[, factornames]
  Ordinals= all[, Ordinalnames]
  combined <- cbind(DFdummies, Ordinals,all_numVar)


# Before returning the processed predictor matrix, I impute all NA values by corresponding mode numbers, determi
ned by the mode function defined as stated above.
  for(j in 1:dim(combined)[2]){
    combined[,j][is.na(combined[,j])]= Mode(combined[,j][!is.na(combined[,j])])
  }
  return(combined)
}
```

Now, with processing method at hand, I implemented feature engineering on the training and test predictor variables together to prevent encountering unseen levels, which is problematic in case of on-hot-encoding traing and test categorical data separately. After preprocessing, separate alldata into training and test data. By now, the 2 sets of data are done with the preprocesing, and ready to be emplyed to train prediction model.

```r
combined=processing(alldata)
train=combined[1:dim(train)[1],]
test=combined[-(1:dim(train)[1]),]
trainprice=log(trainprice)
train=cbind(train,trainprice)
```

Models

Model 1:GBM

Build model, predict SalePrice for Validation set and evaluate the RMSE score. The evaluation codes are not attached here for the limit of the page numbers.

```r
myfit1 = gbm(trainprice ~ . , data = train,
             distribution = "gaussian",
             n.trees = 1000,
             shrinkage = 0.1,
             interaction.depth = 5,
             bag.fraction = 1,
             cv.folds = 5)
```

Export predictions into mysubmission1.txt:

```r
pred <- predict(myfit1, n.trees = which.min(myfit1$cv.error), test)
aaaa=data.frame(PID=testpid,Sale_Price=exp(pred))
write.csv(aaaa,file = "mysubmission1.txt",row.names = F)
```

Models are trained and tested on 10 candidate training and test splits, as in the Project1_test_id.txt file. The accuracy and running time of each are reported as below. (Computer system: 3.2GHz 8GB RAM)

```
Data ID  elapsed-time   RMSE
1          74.46       0.1175802
2          72.13       0.1200228
3          73.41       0.1346245
4          72.94       0.121635
5          76.53       0.1061737
6          73.99       0.1218739
7          76.23       0.1155504
8          85.80       0.1077023
9          73.78       0.1231266
10         71.61       0.1157456
```

Model 2: LASSO

```
start.time = proc.time()
cv.out = cv.glmnet(data.matrix(combined[1:dim(train)[1],]), data.matrix(trainprice), alpha = 1)
best.lam = cv.out$lambda.min
print( proc.time()-start.time )
Ytest.pred = predict(cv.out, s = best.lam, newx = data.matrix(test))
aaaa=data.frame(PID=testpid,Sale_Price=exp(Ytest.pred))
colnames(aaaa)[2]='Sale_Price'
write.csv(aaaa,file = "mysubmission2.txt",row.names = F)
```

The accuracy and running time of each are reported as below. (Computer system: 3.2GHz 8GB RAM)

```
Data ID  elapsed-time   RMSE
1          2.19       0.1559406
2          3.13       0.140281
3          1.92       0.1480856
4          3.36       0.1259131
5          3.09       0.1194714
6          3.39       0.1291439
7          2.26       0.1217073
8          3.76       0.1242258
9          2.33       0.1323659
10         3.66       0.1151544
```