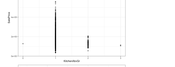
Machine Learning final

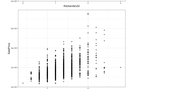
Jade Shi

2022-12-02

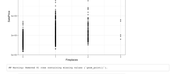
Data Preparation and Exploration

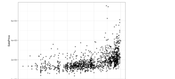
```
Rows: 1,000
Columns: 21
$ checking_status       <chr> "<0", "0<=X<200", "no checking", "<0", "<0", "<0", "0<=X<200", "no checking", "0<=X<200", "<0", "0<=X<200", "no checking", "0<=X<200", "0<=X<200", "0<=X
```

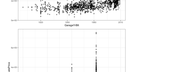


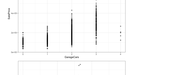


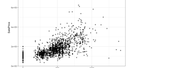

```
/# Warning: Removed 12 rows containing missing values / geom_point:)
```

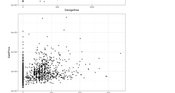


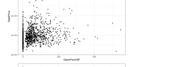


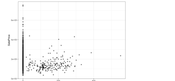


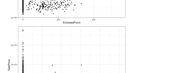


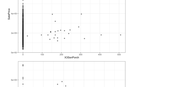










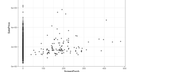


We can conclude that variables like TotRoomsAbvGrd, GrLivArea are significantly related with house sale prices.

Additionally, we will drop outliers of TotRoomsAbvGrd and GrLivArea in train set.

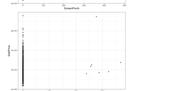
```
outlier1 <- dplyr::filter(train$TotRoomsAbvGrd > 10)$ # prints outlierdataframe # 2000001
outlier2 <- dplyr::filter(train$GrLivArea > 4000)$
outlier3 <- dplyr::filter(train$TotRoomsAbvGrd > 10 & grlivarea > 4000)$ # 2000001
dplyr::bind_rows(outlier1,outlier2,outlier3)
```

```
/# Neighborhood
ggplot(data=train, mapping = aes(x=Neighborhood, y = SalesPrice))
+ geom_boxplot()
+ theme(axis.text.x = element_text(angle = 90))
+ ylim(0,500000)
+ theme(axis.text.x = element_text(size = 7))
+ theme(axis.text.y = element_text(size = 7))
+ theme(axis.title = element_text(size = 7))
```



It's obvious that average house prices and house price distributions for each neighborhood are very different.

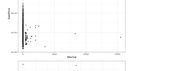
```
/# ExterQual
ggplot(data=train, mapping = aes(x=ExterQual, y = SalesPrice))
+ geom_boxplot()
+ ylim(0,500000)
+ theme(axis.text.x = element_text(size = 7))
+ theme(axis.text.y = element_text(size = 7))
+ theme(axis.title = element_text(size = 7))
```



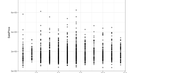

```
/# BsmtQual
ggplot(data=train, mapping = aes(x=BsmtQual, y = SalesPrice))
+ geom_boxplot()
+ ylim(0,500000)
+ theme(axis.text.x = element_text(size = 7))
+ theme(axis.text.y = element_text(size = 7))
+ theme(axis.title = element_text(size = 7))
```


We can assume that ExterQual and BsmtQual are correlated with house sale prices, while the BsmtFinType1 is slightly correlated with house sale prices.

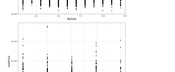
```
/# Corr: Numerical vs Numerical
NumVars <- which(sapply(train,FUN=function(x) is.numeric(x)))
train_numeric <- train[,NumVars]
cor_numvar <- cor(train_numeric,use="pairwise.complete.obs")
cor_sorted <- as.matrix(sort(cor[,'SalesPrice'],decreasing=TRUE))
corhigh <- names(which(apply(abs(cor_sorted)>=0.5,1,sum)>1))
cor = cor(train[,corhigh])
corrplot::corrplot.mixed(cor,tl.col="black", tl.pos = "lt", tl.cex = 7,cl.cex = 7)
```



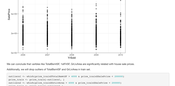

```
/# Corr: Numerical vs Numerical
NumVars <- which(sapply(train,FUN=function(x) is.numeric(x)))
train_numeric <- train[,NumVars]
cor_numvar <- cor(train_numeric,use="pairwise.complete.obs")
cor_sorted <- as.matrix(sort(cor[,'SalesPrice'],decreasing=TRUE))
corlow <- names(which(apply(abs(cor_sorted)<=0.5,1,sum)>1))
cor = cor(train[,corlow])
corrplot::corrplot.mixed(cor,tl.col="black", tl.pos = "lt", tl.cex = 7,cl.cex = 7)
```




```
/# Corr: Numerical vs Numerical
NumVars <- which(sapply(train,FUN=function(x) is.numeric(x)))
train_numeric <- train[,NumVars]
cor_numvar <- cor(train_numeric,use="pairwise.complete.obs")
cor_sorted <- as.matrix(sort(cor[,'SalesPrice'],decreasing=TRUE))
corhigh <- names(which(apply(abs(cor_sorted)>=0.5,1,sum)>1))
cor = cor(train[,corhigh])
corrplot::corrplot.mixed(cor,tl.col="black", tl.pos = "lt", tl.cex = 7,cl.cex = 7)
```




```
/# Corr: Numerical vs Numerical
NumVars <- which(sapply(train,FUN=function(x) is.numeric(x)))
train_numeric <- train[,NumVars]
cor_numvar <- cor(train_numeric,use="pairwise.complete.obs")
cor_sorted <- as.matrix(sort(cor[,'SalesPrice'],decreasing=TRUE))
corlow <- names(which(apply(abs(cor_sorted)<=0.5,1,sum)>1))
cor = cor(train[,corlow])
corrplot::corrplot.mixed(cor,tl.col="black", tl.pos = "lt", tl.cex = 7,cl.cex = 7)
```


MSSubClass
BsmtFinSF2
BsmtUnfSF
BedroomAbvGr
ScreenPorch
PoolArea
MoSold
3SsnPorch
BsmtFinSF2
BsmtHalfBath
MiscVal
LowQualFinSF
YrSold
OverallCond
MSSubClass
EnclosedPorch
KitchenAbvGr
1.0
0.8
0.6
0.4
0.2

```
zero <- c('GarageYrBlt', 'MasVnrArea', 'BsmtFullBath', 'BsmtHalfBath', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', '
GarageCars', 'GarageArea', 'TotalBsmtSF')
price_train[zero][is.na(price_train[zero])] <- 0
price_test[zero][is.na(price_test[zero])] <- 0
```

For other columns missing values, we can fill out NA values by the mode value.

```
#MSZoning #Exterior1st #Exterior2nd #SaleType #KitchenQual #Electrical
mode <- c('MSZoning', 'Exterior1st', 'Exterior2nd', 'SaleType', 'KitchenQual', 'Electrical')
for (col in mode) {
  mode_value <- names(sort(table(price_train[[col]]), decreasing = TRUE))[1]
  price_train[[col]][is.na(price_train[[col]])] <- mode_value
  price_test[[col]][is.na(price_test[[col]])] <- mode_value
}
```

```
# LotFrontage
price_train$LotFrontage <- ave(price_train$LotFrontage, price_train$Neighborhood, FUN = function(x) {
  x[is.na(x)] <- median(x, na.rm = TRUE)
  x
})
missing_indices <- which(is.na(price_test$LotFrontage))
for (ind in missing_indices) {
  x <- price_test$Neighborhood[ind]
  price_test$LotFrontage[ind] <- median(price_train$LotFrontage[price_train$Neighborhood == x], na.rm = TRUE)
}
```

```
# Drop the column Utilities
price_train <- price_train[, !(names(price_train) %in%c('Utilities'))]
price_test <- price_test[, !(names(price_test) %in% c('Utilities'))]
```

```
# Convert categorical variables to numeric values
cate_features <- cate_features[cate_features != 'Utilities']
library(dplyr)
for (col in cate_features) {
  price_train[[col]] <- as.character(price_train[[col]])
  price_test[[col]] <- as.character(price_test[[col]])
}

label <- c('Street', 'Alley', 'LotShape', 'LandContour', 'LandSlope', 'HouseStyle', 'RoofMatl', 'Exterior1st', 'E
xterior2nd', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'Bsm
tFinType2', 'HeatingQC', 'CentralAir', 'KitchenQual', 'Functional', 'FireplaceQu', 'GarageFinish', 'GarageQual',
'GarageCond', 'PavedDrive', 'PoolQC', 'Fence')
for (col in label) {
  value_train <- unique(price_train[[col]])
  value_test <- unique(price_test[[col]])
  value_list <- unique(c(value_train, value_test))

  price_train[[col]] <- as.integer(factor(price_train[[col]], levels = value_list))
  price_test[[col]] <- as.integer(factor(price_test[[col]], levels = value_list))
}
```

```
combined <- bind_rows(price_train[, !names(price_train) %in% "SalePrice"], price_test) %>%
  mutate_if(is.factor, as.character)
combined <- model.matrix(~ . - 1, data = combined)
```

```
# Save the cleaned train set and test set
trainset <- as.data.frame(combined[1:1458, ])
y <- price_train$SalePrice
testset <- as.data.frame(combined[1459:nrow(combined), ])

cat("The shape of training data:", dim(trainset), "\n")
```

```
## The shape of training data: 1458 156
```

```
cat("The shape of testing data:", dim(testset), "\n")
```

```
## The shape of testing data: 1459 156
```

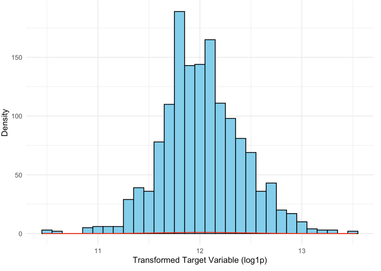
```
# Scale y
y <- log1p(y)

skewness <- skewness(y)
kurtosis <- kurtosis(y)
print(paste('Skewness of target:', skewness))
```

```
## [1] "Skewness of target: 0.121454642681859"
```

```
print(paste('Kurtosis of target:', kurtosis))
```

```
## [1] "Kurtosis of target: 3.79788125830084"
```

```
ggplot(data = data.frame(y), aes(x = y)) +
  geom_histogram(binwidth = 0.1, fill = "skyblue", color = "black") +
  stat_function(fun = dnorm, args = list(mean = mean(y), sd = sd(y)), color = "red") +
  labs(x = "Transformed Target Variable (log1p)", y = "Density") +
  theme_minimal()
```



Model Fitting and Predictions

```
set.seed(1)
lasso <-trainControl(method="cv", number=10)
lassoGrid <- expand.grid(alpha = 1, lambda = seq(0.001,0.1,by = 0.0005))

lasso_model <- train(x=trainset, y=y, method='glmnet', trControl= lasso, tuneGrid=lassoGrid)
lasso_model$bestTune
```

```
##   alpha lambda
## 3     1  0.002
```

```
min(lasso_model$results$RMSE)
```

```
## [1] 0.1128931
```

```
# Lasso Model
lasso_preds <- predict(lasso_model, testset)
predictions_lasso <- exp(lasso_preds)
head(predictions_lasso)
```

```
##     1459     1460     1461     1462     1463     1464
## 120372.6 157824.9 179536.9 199082.8 201592.1 173252.4
```

```
xgb_grid = expand.grid(
nrounds = 1000,
eta = c(0.1, 0.05, 0.01),
max_depth = c(2, 3, 4, 5, 6),
gamma = 0,
colsample_bytree=1,
min_child_weight=c(1, 2, 3, 4 ,5),
subsample=1
)
label_train <- y
xtrain <- xgb.DMatrix(data = as.matrix(trainset), label= label_train)
xtest <- xgb.DMatrix(data = as.matrix(testset))
```

```
default_param<-list(
        objective = "reg:linear",
        booster = "gbtree",
        eta=0.05, #default = 0.3
        gamma=0,
        max_depth=3, #default=6
        min_child_weight=4, #default=1
        subsample=1,
        colsample_bytree=1
)
```

```
xgbcv <- xgb.cv( params = default_param, data = xtrain, nrounds = 500, nfold = 10, showwd = T, stratified = T, pr
int_every_n = 40, early_stopping_rounds = 10, maximize = F)
```

```
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [08:02:53] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
## [1]  train-rmse:10.955542+0.003302   test-rmse:10.955487+0.031407
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [41] train-rmse:1.428767+0.000447    test-rmse:1.429125+0.018324
## [81] train-rmse:0.221934+0.000915    test-rmse:0.233382+0.013341
## [121]    train-rmse:0.105508+0.001526    test-rmse:0.131550+0.018447
## [161]    train-rmse:0.093257+0.001423    test-rmse:0.123956+0.018366
## [201]    train-rmse:0.086835+0.001460    test-rmse:0.120922+0.017849
## [241]    train-rmse:0.082279+0.001433    test-rmse:0.119355+0.017541
## [281]    train-rmse:0.078674+0.001434    test-rmse:0.118477+0.017221
## [321]    train-rmse:0.075504+0.001404    test-rmse:0.117853+0.017015
## [361]    train-rmse:0.072574+0.001399    test-rmse:0.117558+0.016914
## Stopping. Best iteration:
## [360]    train-rmse:0.072654+0.001390    test-rmse:0.117531+0.016917
```

```
xgb_mod <- xgb.train(data = xtrain, params=default_param, nrounds = 454)
```

```
## [08:03:05] WARNING: src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarede
rror.
```

```
XGBpred <- predict(xgb_mod, xtest)
predictions_XGB <- exp(XGBpred)
head(predictions_XGB)
```

```
## [1] 120827.7 159677.5 188426.5 191070.1 191908.7 175801.6
```

```
#Averaging Predictions
test_id <- seq(1461, 2919)
average_y <- data.frame(Id = test_id, SalePrice = (predictions_lasso * 2 + predictions_XGB) / 3)
head(average_y)
```

```
##        Id SalePrice
## 1459 1461  120524.3
## 1460 1462  158442.5
## 1461 1463  182500.1
## 1462 1464  196411.9
## 1463 1465  198364.3
## 1464 1466  174102.2
```

```
# export data
write.csv(average_y, file = "average_predictions.csv", row.names = FALSE)
```