ML_FinalProject

Group

2023-04-23

```
Sys.setenv(LANG = "en")
```

#1. library

```
library(r02pro)
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.2.3

## Warning: package 'ggplot2' was built under R version 4.2.3

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'tidyr' was built under R version 4.2.3

## Warning: package 'readr' was built under R version 4.2.3

## Warning: package 'purrr' was built under R version 4.2.3

## Warning: package 'stringr' was built under R version 4.2.3

## Warning: package 'forcats' was built under R version 4.2.3

## Warning: package 'lubridate' was built under R version 4.2.3
```

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.2.3
```

```
library(tree)
library(ISLR)
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.2.3
```

```
library(pROC)
library(ggplot2)
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 4.2.3
```

```
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.2.3
```

```
library(tfdatasets)
```

```
## Warning: package 'tfdatasets' was built under R version 4.2.3
```

```
library(dplyr)
```

1. import data

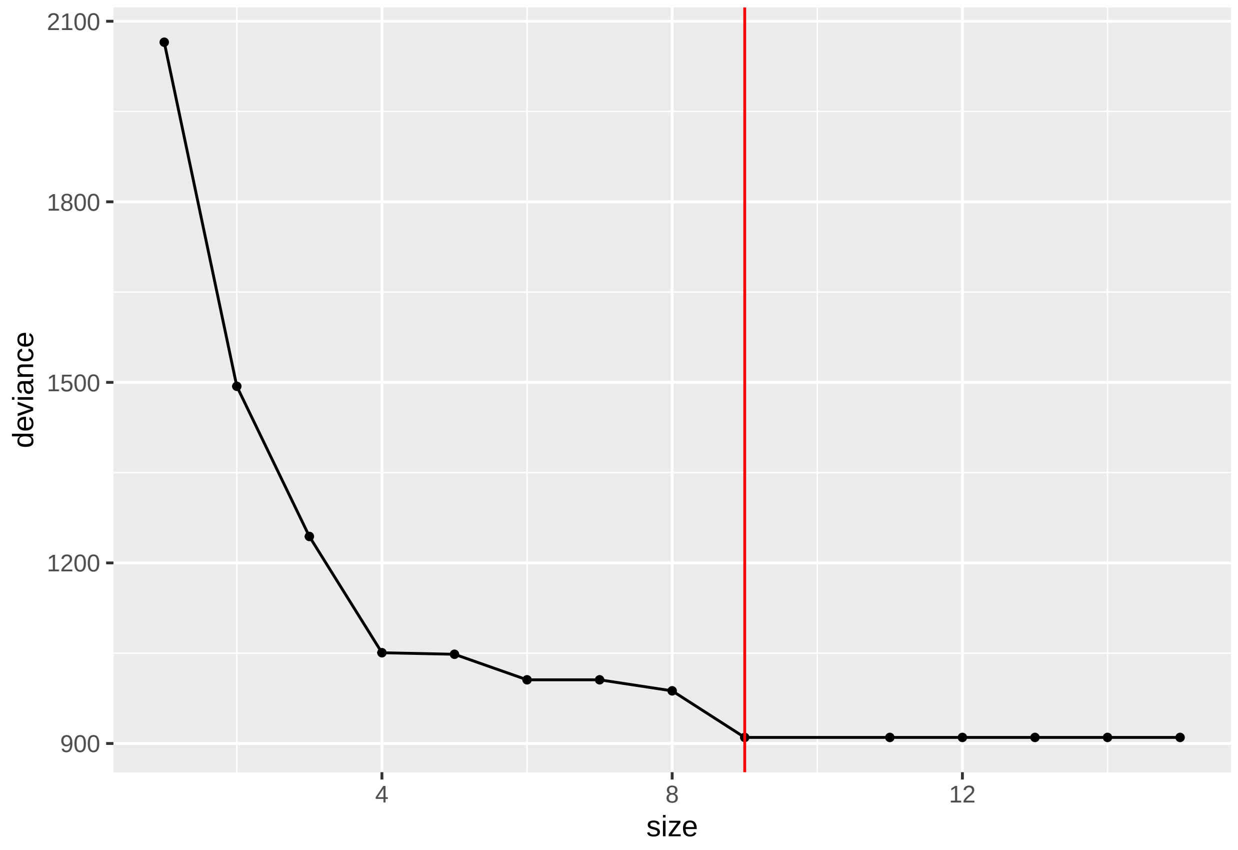
```
California_Houses <- read.csv("California_Houses.csv")
set.seed(123)
California_Houses_reg <- California_Houses %>% dplyr::select(-Latitude, -Longitude) %>% na.omit()

#create training index that is 10% of original size
train_idx <- sample(nrow(California_Houses_reg), nrow(California_Houses_reg) * 0.1)
train_df <- California_Houses_reg[train_idx, ]
test_df <- California_Houses_reg[-train_idx, ]
```

2. Scale all features into the interval $[0, 1]$.

```
#standardization
std_fit <- preProcess(train_df, method = "scale")
train_std <- predict(std_fit, newdata = train_df)
std_fit_test <- preProcess(test_df, method = "scale")
test_std <- predict(std_fit_test, newdata = test_df)
```

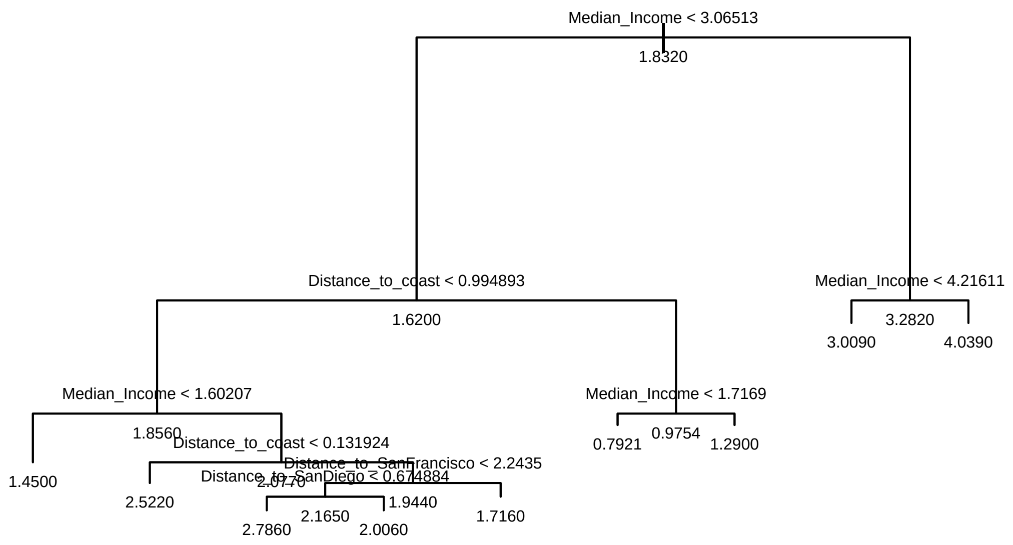
3. Decision Tree

```r
#decision tree with CV pruning.

my_control <- tree.control(nrow(train_std),mincut=50, minsize = 200, mindev = 0)
fit <- tree(Median_House_Value ~ . ,control = my_control,data = train_std)
set.seed(0)
cv.fit <- cv.tree(fit)
cv.fit_df <- data.frame(size = cv.fit$size, deviance = cv.fit$dev)
least_dev <- min(cv.fit$dev)
best_size <- min(cv.fit$size[cv.fit$dev == least_dev])

#plot best tree size
ggplot(cv.fit_df, mapping = aes(x = size, y = deviance)) +
geom_point(size = 1) +
geom_line() +
geom_vline(xintercept = best_size, col = "red")
```



```r
#fit tree with the best size
fit.tree<-prune.tree(fit, best = best_size)

#training error and test error
pred_fit <- predict(fit.tree, newdata = train_std)
train_error_dt<- mean((pred_fit - train_std$Median_House_Value)^2)


pred_fit_te<- predict(fit.tree,newdata = test_std)
```

```r
test_error_dt<- mean((pred_fit_te - test_std$Median_House_Value)^2)

#plot the tree
plot(fit.tree)
text(fit.tree,  all = TRUE, cex = 0.5)
```



```r
#Evaluation
summary(fit.tree)
```

```
##
## Regression tree:
## snip.tree(tree = fit, nodes = c(10L, 39L, 77L, 8L))
## Variables actually used in tree construction:
## [1] "Median_Income"            "Distance_to_coast"
## [3] "Distance_to_SanFrancisco" "Distance_to_SanDiego"
## Number of terminal nodes:  9
## Residual mean deviance:  0.4069 = 836.1 / 2055
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.15800 -0.36700 -0.07141  0.00000  0.30940  3.14600
```

4.Random Forest

```
set.seed(1)
#fit the model
randomforest <- randomForest(Median_House_Value ~ .,data = train_std,importance = TRUE)
pred_rf<- predict(randomforest, newdata =train_std)

#test and training error
train_error_rf<- mean((pred_rf - train_std$Median_House_Value)^2)
train_error_rf
```

```
## [1] 0.04510056
```

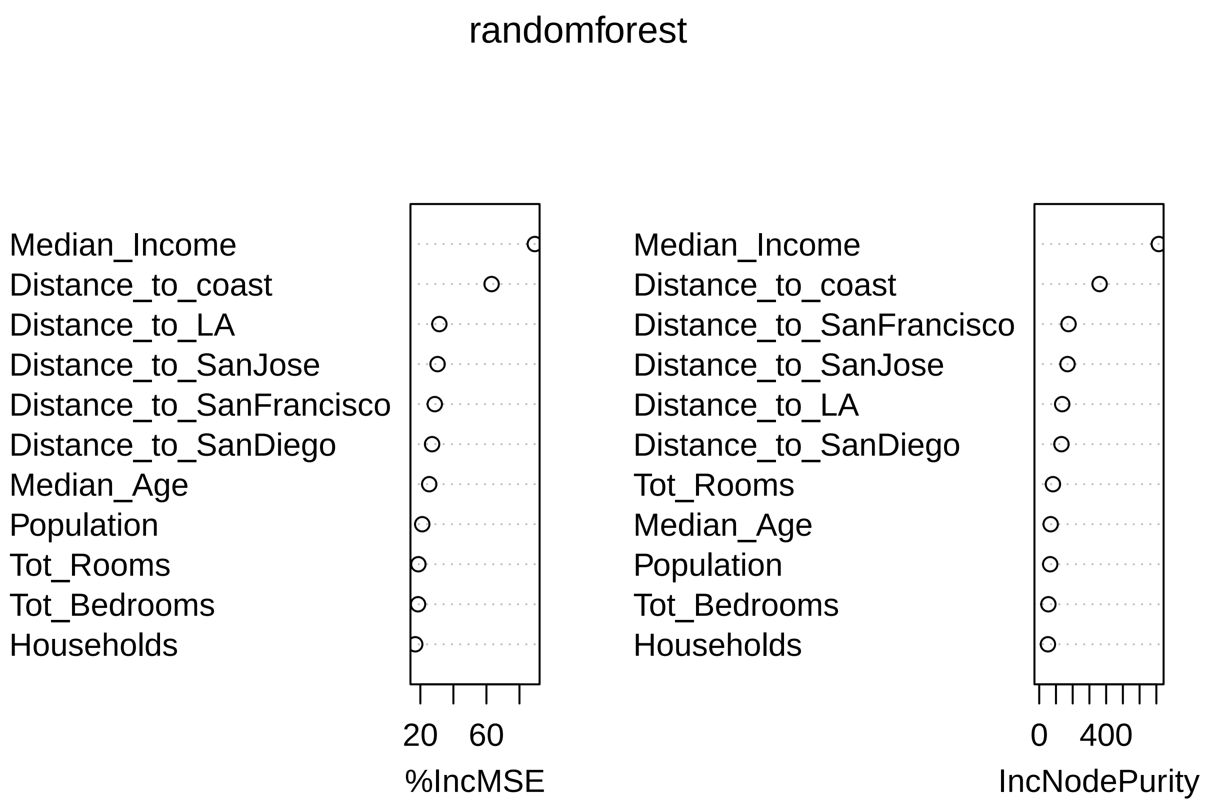
```
pred_rf_te<- predict(randomforest, newdata =test_std)
test_error_rf<- mean((pred_rf_te - test_std$Median_House_Value)^2)
test_error_rf
```

```
## [1] 0.2421523
```

```
#Evaluation
importance(randomforest)
```

```
##                            %IncMSE IncNodePurity
## Median_Income             89.29851     714.79316
## Median_Age                25.38105      68.34348
## Tot_Rooms                 18.79392      81.92050
## Tot_Bedrooms              18.59879      54.58898
## Population                21.15065      65.50398
## Households                16.91128      51.95155
## Distance_to_coast         63.13495     360.91513
## Distance_to_LA            31.51166     137.37541
## Distance_to_SanDiego      27.11946     132.87531
## Distance_to_SanJose       30.43295     169.03874
## Distance_to_SanFrancisco  28.74694     175.21282
```

```
varImpPlot(randomforest)
```

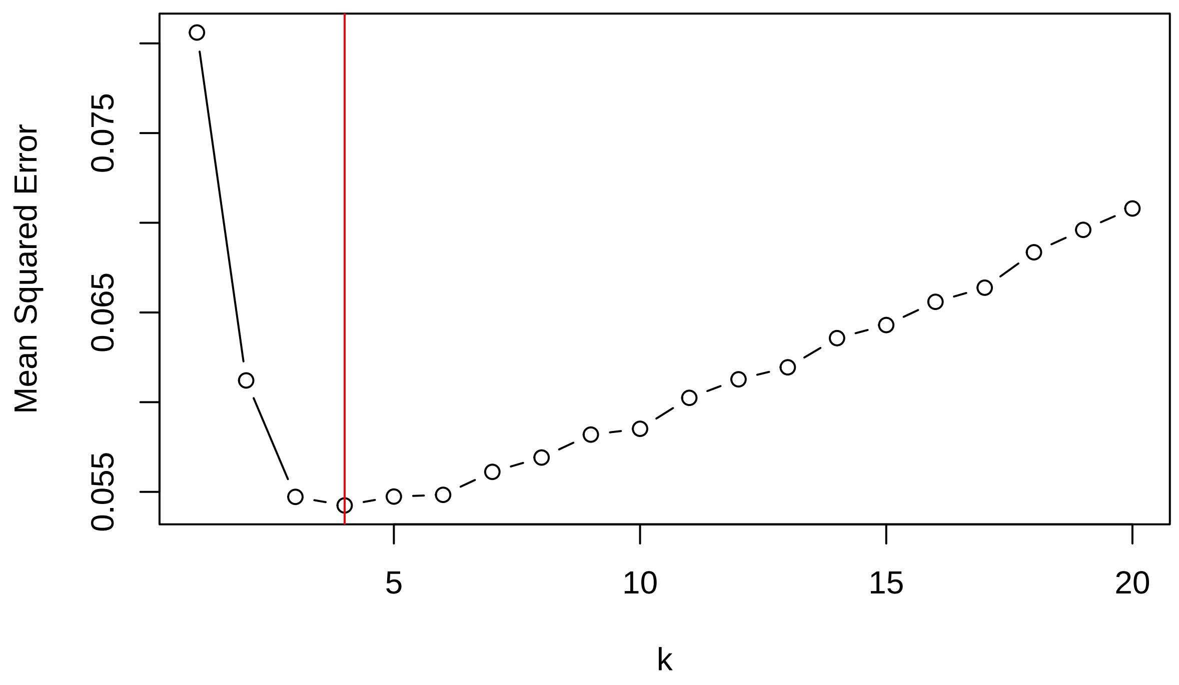


5. Knn regression

```
#make training set a data frame
df <- as.data.frame(train_std)

#make median housing value numeric
df$Median_House_Value <- as.numeric(df$Median_House_Value)


train_std$Median_House_Value <- df$Median_House_Value

#setup k sequence
k_seq <- seq(from = 1, to = 20, by = 1)

#determine CV error
set.seed(123)
k_fold <- createFolds(train_std$Median_House_Value, k = 10)
CV_error <- sapply(k_seq, function(k) {
  mean(sapply(seq_along(k_fold), function(i) {

    train <- train_std[-k_fold[[i]],]
    test <- train_std[k_fold[[i]],]
    fit_knn <- knnreg(train[, -12], train$Median_House_Value, k = k)

    pred <- predict(fit_knn, newdata = test[, -12])
    mean((pred - test$Median_House_Value)^2)
  }))
})

#choose the best k using least CV error
best_k <- k_seq[which.min(CV_error)]
best_k
```

```
## [1] 4
```

```
#visualization of the best k
plot(k_seq, CV_error, type = "b", xlab = "k", ylab = "Mean Squared Error")
abline(v = best_k, col = "red")
```



```
#fit the model
  pred_knn <- knnreg(Median_House_Value ~ ., data = train_std, k = 4)

#find test and training error
  y_train_hat_knn <- predict(pred_knn, newdata = train_std)
 train_error_knn<- mean((train_std$Median_House_Value - y_train_hat_knn)^2)
 train_error_knn
```

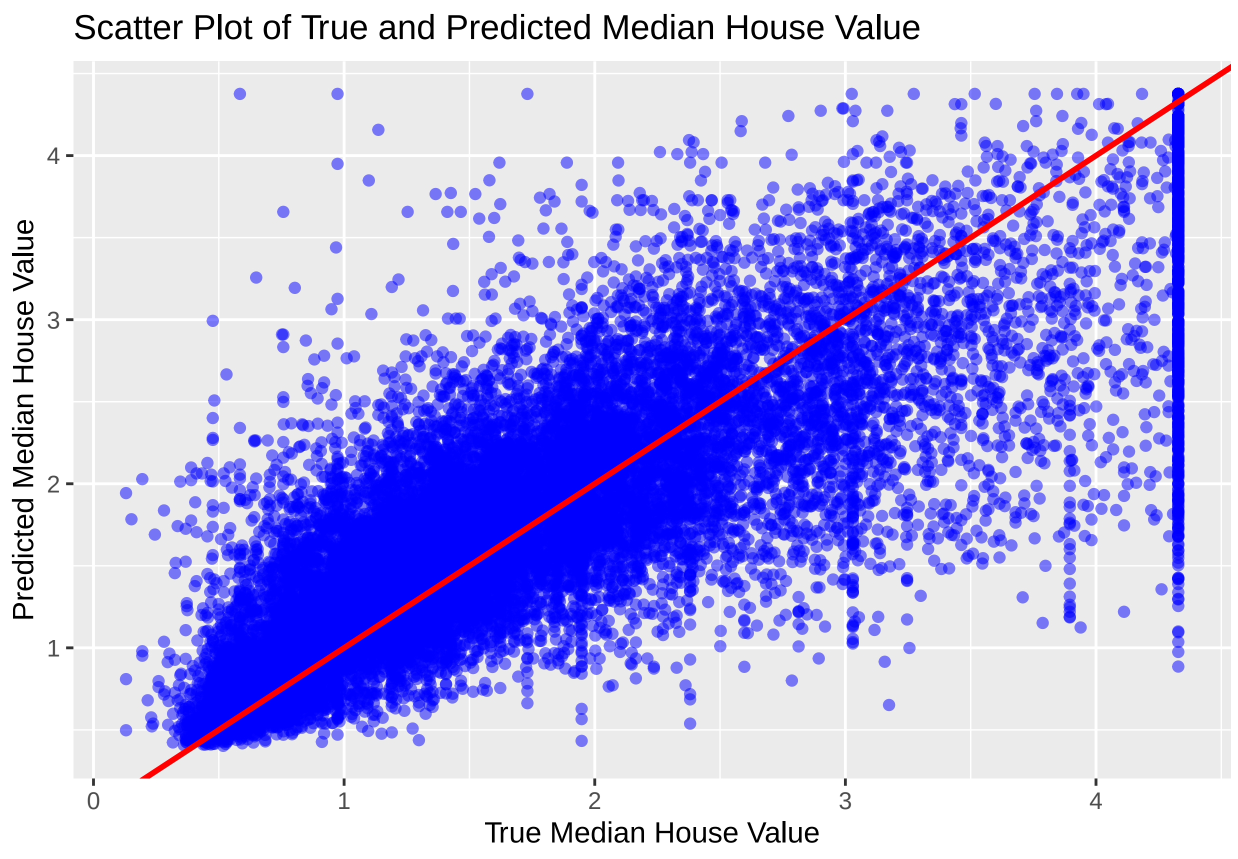
```
## [1] 0.2147305
```

```
 y_test_hat_knn <- predict(pred_knn, newdata = test_std)
test_error_knn<-  mean((test_std$Median_House_Value - y_test_hat_knn)^2)
 test_error_knn
```

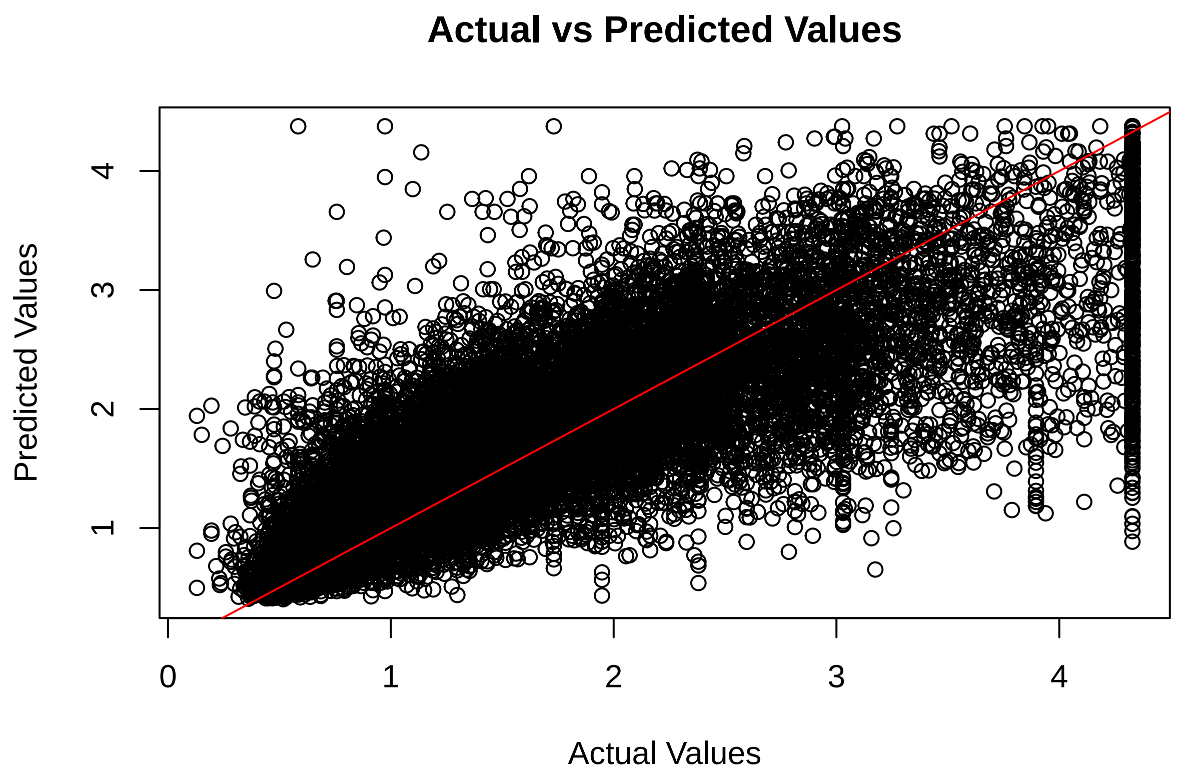
```
## [1] 0.356129
```

```
#visualization of predicted with true value
 ggplot(data = test_std, aes(x = Median_House_Value, y = y_test_hat_knn)) +
 geom_point(color = "blue", alpha = 0.5) +
 geom_abline(color = "red", size = 1) +
 xlab("True Median House Value") +
 ylab("Predicted Median House Value") +
 ggtitle("Scatter Plot of True and Predicted Median House Value")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```
  plot(test_std$Median_House_Value, y_test_hat_knn, main = "Actual vs Predicted Values", xlab = "Actual
abline(0, 1, col = "red")
```



6. Deep learning

Scale x and y into the interval $[0, 1]$ seperately.

```
train_x_std <- train_std %>% dplyr::select(-Median_House_Value)
train_y_std <- train_std$Median_House_Value
test_x_std <- test_std %>% dplyr::select(-Median_House_Value)
test_y_std <- test_std$Median_House_Value
 train_x_mat<- as.matrix(train_x_std)
test_x_mat <- as.matrix(test_x_std)
```

1. Setup the layers

```
ptrain <- ncol(train_x_std)
model <- keras_model_sequential()
model %>%
  layer_dense(
    units = 64,
    activation = 'relu',
    input_shape = c(ptrain)
  )   %>%
  layer_dense(units = 1, activation = 'linear')
summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_1 (Dense)                    (None, 64)                       768
##  dense (Dense)                      (None, 1)                        65
## ================================================================================
## Total params: 833
## Trainable params: 833
## Non-trainable params: 0
## ________________________________________________________________________________
```

Improve the model with 2 additional hidden layers

```
ptrain <- ncol(train_x_std)
model <- keras_model_sequential()
model %>%
  layer_dense(
    units = 64,
    activation = 'relu',
    input_shape = c(ptrain)
  ) %>%
  layer_dense(
    units = 64,
    activation = 'relu'
  ) %>%
  layer_dense(
    units = 64,
    activation = 'relu'
  ) %>%
  layer_dense(units = 1, activation = 'linear')
summary(model)
```

```
## Model: "sequential_1"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_5 (Dense)                    (None, 64)                       768
##  dense_4 (Dense)                    (None, 64)                       4160
##  dense_3 (Dense)                    (None, 64)                       4160
##  dense_2 (Dense)                    (None, 1)                        65
## ================================================================================
## Total params: 9,153
## Trainable params: 9,153
## Non-trainable params: 0
## ________________________________________________________________________________
```

2. Compile the model

```
model %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_rmsprop(),
  metrics = c('mse')
)
```

3. Train the model

```
history <- model %>% fit(
  train_x_mat,
  train_y_std,
  epochs = 10,
  batch_size = 16,
  validation_split = 0.1
)
```

4. Evaluate the accuracy

```
score <- model %>% evaluate(test_x_mat, test_y_std, verbose = 0)
cat('Test loss:', score["loss"], "\n")
```

```
## Test loss: 0.3856033
```

5. Make predictions

```
y_hat <- model %>% predict(train_x_mat)
y_pred <- model %>% predict(test_x_mat)
train_error_dl<-mean((y_hat - train_y_std)^2)
test_error_dl<-mean((y_pred - test_y_std)^2)
train_error_dl
```

```
## [1] 0.311725
```

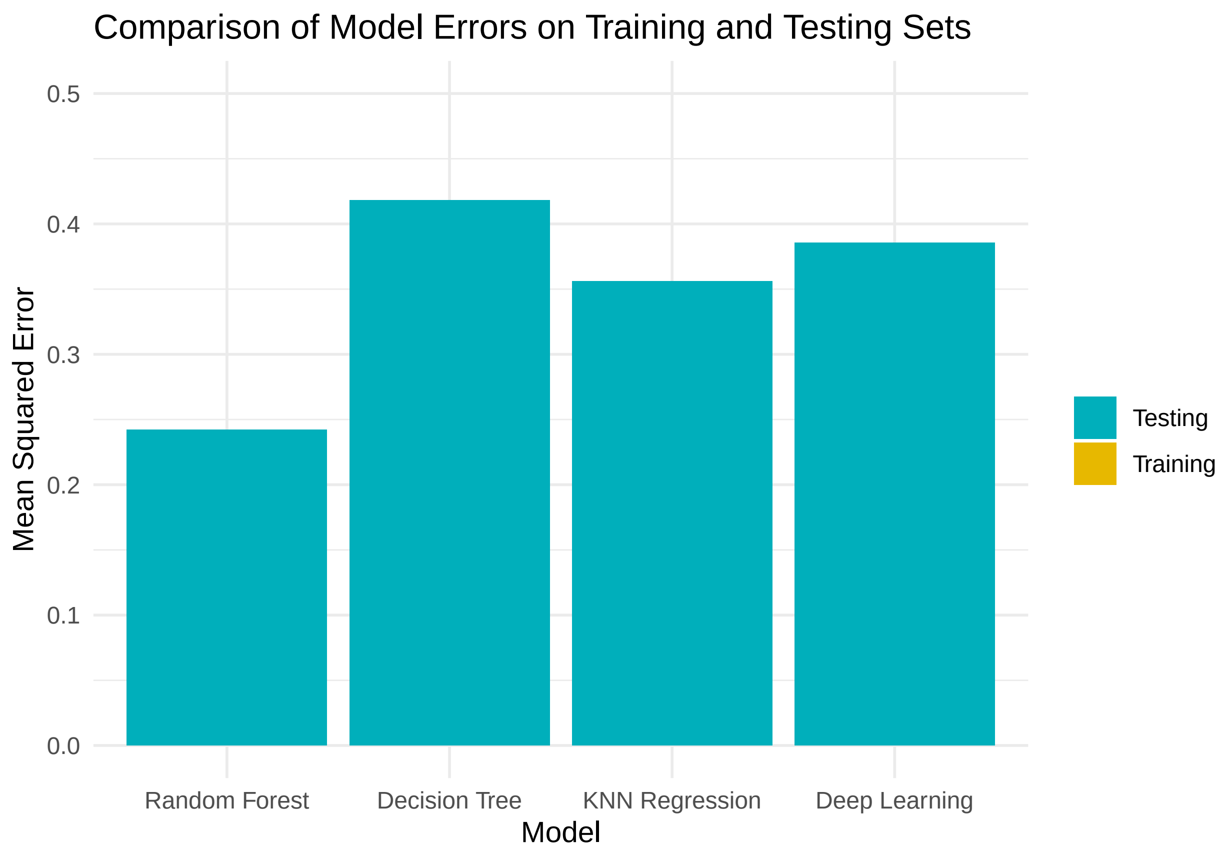
```
test_error_dl
```

```
## [1] 0.3856032
```

```
# create a data frame
model_names <- c("Random Forest", "Decision Tree", "KNN Regression","Deep Learning")
train_errors <- c(train_error_rf, train_error_dt, train_error_knn,train_error_dl)
test_errors <- c(test_error_rf, test_error_dt, test_error_knn,test_error_dl)
errors_df <- data.frame(model_names, train_errors, test_errors)
errors_df
```

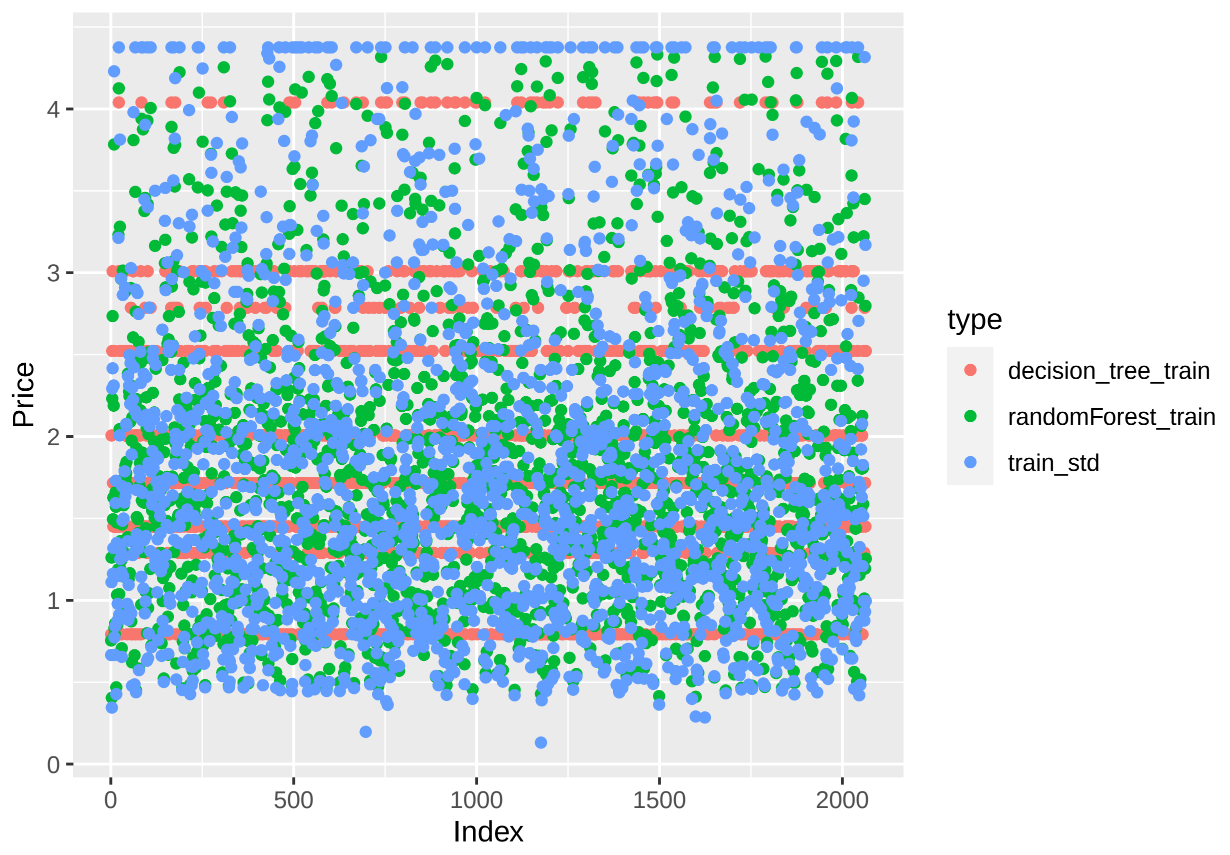
```
##      model_names train_errors test_errors
## 1  Random Forest   0.04510056   0.2421523
## 2  Decision Tree   0.40508671   0.4182491
## 3 KNN Regression   0.21473050   0.3561290
## 4  Deep Learning   0.31172498   0.3856032
```

```r
# make visualization
ggplot(errors_df, aes(x = model_names, y = train_errors, fill = "Training")) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_bar(aes(y = test_errors, fill = "Testing"), stat = "identity", position = "dodge") +
  scale_fill_manual(values = c("#00AFBB", "#E7B800")) +
  scale_y_continuous(limits=c(0,0.5)) +
  scale_x_discrete(limits=c("Random Forest", "Decision Tree", "KNN Regression","Deep Learning")) +
  labs(x = "Model", y = "Mean Squared Error", fill = "") +
  ggtitle("Comparison of Model Errors on Training and Testing Sets") +
  theme_minimal()
```



7. visualization

```r
index<- seq_len(nrow(train_std))
cp<- rbind(data.frame(Index = index, Price = pred_fit, type = "decision_tree_train"),
      data.frame(Index = index, Price = pred_rf, type = "randomForest_train"),
      data.frame(Index = index, Price = train_std$Median_House_Value, type = "train_std"))
ggplot(cp, mapping = aes(x = Index, y = Price, color = type)) +
geom_point()
```



8. test and training error

```
#make a data frame that contains all test and training errors.
model_names <- c('KNN', 'Random Forest', 'Decision Tree', 'Deep Learning')
test_e <- c(test_error_knn, test_error_rf, test_error_dt, test_error_dl)
training_e <- c( train_error_knn, train_error_rf, train_error_dt, train_error_dl)
results_df <- data.frame(Model = model_names, test_error = test_e, training_error = training_e)

# Print the results
print(results_df)
```

```
##           Model test_error training_error
## 1           KNN  0.3561290     0.21473050
## 2 Random Forest  0.2421523     0.04510056
## 3 Decision Tree  0.4182491     0.40508671
## 4 Deep Learning  0.3856032     0.31172498
```