ISYE 7406: Homework # 3 Report

Introduction:

The objective of this analysis is to analyze a car dataset and predict whether a vehicle has high or low fuel efficiency, which is measured by miles per gallon (MPG), based on several other vehicle features. The dataset used in this analysis comes from the UCI Machine Learning Repository. The original dataset consists of 398 rows, each representing a different vehicle, and 9 columns representing various car attributes. The original dataset contained missing values, which were removed in the data cleaning process, along with irrelevant columns. The new dataset contains 392 observations and 8 columns.

For this analysis I developed multiple different classification models and compared their performance to determine the optimal model. The target variable is Miles per gallon (MPG), which is a unit used to measure fuel efficiency. A higher MPG value indicates better gas mileage. Vehicles with high MPG can travel a greater distance on less fuel, saving money on fuel costs while also reducing CO2 emissions. Vehicles with low MPG are less fuel-efficient, requiring more fuel and producing higher emissions.By evaluating models based on test error and variance, I aim to identify the most effective classification approach for predicting fuel efficiency.

Exploratory Data Analysis (EDA):

Before modeling, I conducted an exploratory data analysis (EDA) to examine the relationships between various vehicle attributes and fuel efficiency. The cleaned dataset contains no missing values, ensuring that all observations are complete for analysis. To prepare the target variable MPG for classification models, I converted MPG into a categorical variable, labeling observations as either "High" or "Low" fuel efficiency based on the median MPG value. Many classification algorithms require the target variable to be categorical.

The target variable is evenly distributed, with 50% of the observations classified as high MPG and 50% as low MPG. Each class has 196 observations. The target variable is balanced, so there is no need for balancing techniques. This balance ensures fair training and evaluation without bias towards one class over the other.

To explore the relationships between fuel efficiency and other vehicle attributes, I generated boxplots comparing each independent variable against MPG categories. The boxplots of MPG vs the independent variables show whether significant differences exist between the classes. All of the independent variables except for acceleration appear to show a significant difference in medians between the high and low MPG categories. Cylinder, weight, year and origin show the most pronounced differences between fuel efficiency classes. Acceleration and model year are higher in high-MPG cars, meaning vehicles with newer model years or higher acceleration have higher fuel efficiency. Vehicles with higher weight, horsepower, cylinders, or displacement tend to have lower fuel efficiency. This aligns with the understanding that larger, more powerful engines consume more fuel.

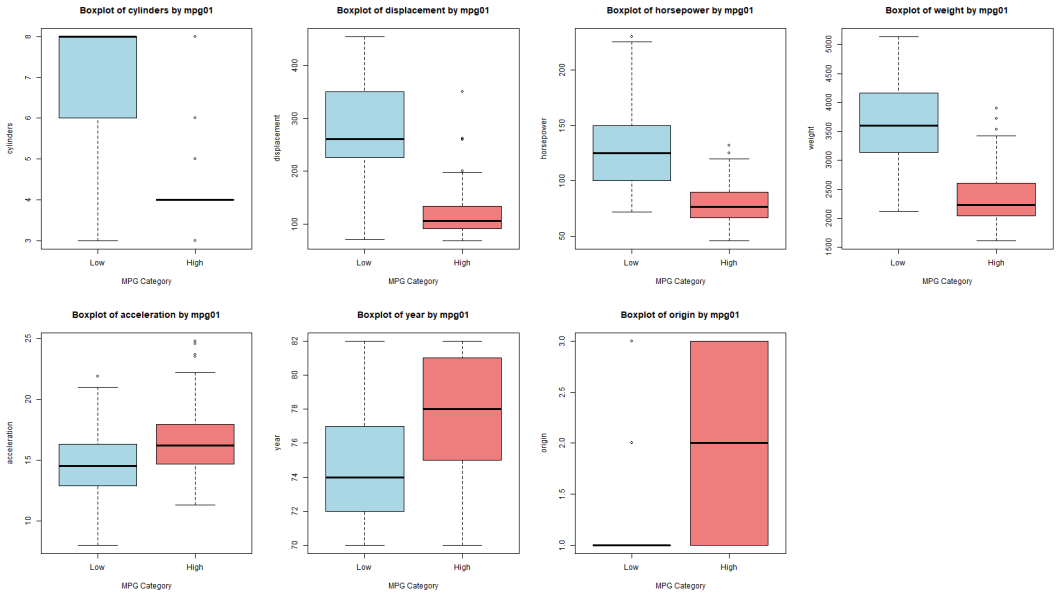

Figure 1: Boxplots of the target vs the predictor variables.

Correlations: For the correlation analysis, I constructed a correlation matrix. MPG had strong negative correlations with predictors like cylinders (-0.76), displacement (-0.75), horsepower (-0.67), and weight(-0.76). Additionally, some predictor variables have strong correlations among each other, such as, cylinders with displacement(0.95), cylinders with horsepower (0.84), and cylinders with weight (0.90). The features most strongly correlated with fuel efficiency include weight, displacement, horsepower, and cylinders. These variables may play a critical role in distinguishing between high and low fuel efficiency vehicles.

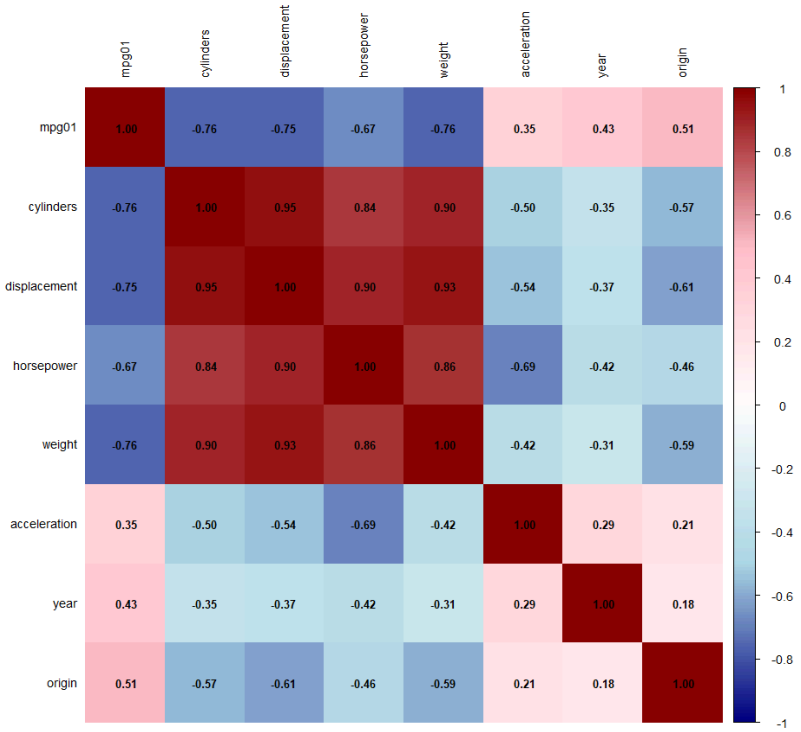

Figure 2: Correlation Matrix with self correlations on the diagonal. A coefficient of 1 indicates a strong positive relationship, 0 indicates no relationship, and -1 indicates a strong negative relationship. Blue represents negative relationships and red represents positive relationships.

Methodology:

I built several different classification models using different approaches. I converted the target variable, MPG, into a binary variable, labeling values below the median as "Low" and values above the median as "High." This classification provides better clarity when predicting whether a vehicle has good or poor fuel efficiency. After labeling, I converted the variable into factor levels using the factor() function in R, ensuring compatibility with classification algorithms that require categorical target variables.

The dataset was split into training and testing sets. The models using Monte Carlo cross-validation with 100 iterations, were split using random sampling, where 10% of the data was randomly selected as the test set in each iteration.

The Linear Discriminant Analysis (LDA) model was implemented first. LDA is a supervised learning algorithm used for classification. It finds a linear combination of features that separates different classes optimally. It assumes classes follow a Gaussian (normal) distribution and share a common covariance matrix. Because the classes share a common covariance matrix in LDA, it results in a linear decision boundary. The lda() function from the MASS library was used to implement the model.

The next model implemented was the Quadratic Discriminant Analysis (QDA) model. The QDA algorithm is able to handle nonlinear and complex relationships. Unlike LDA where each class shares a common covariance matrix, QDA allows each class to have its own covariance matrix, which results in quadratic decision boundaries. Just like LDA, it has the assumption of normal distribution of classes. The qda() function in R from the MASS library was used.

Naive Bayes was the third model implemented in the analysis. Naive Bayes is a probabilistic classifier that assumes all features are independent, unlike LDA and QDA. There may be some limitations with this assumption, because in the real-world this is not always the case. There are different variations of Naive Bayes for different distributions like Gaussian, Bernoli, multinomial, and Poisson. The naiveBayes() function from the e1071 library in R was used.

Logistic Regression was implemented next. Logistic Regression is also a supervised learning algorithm used for classification. It uses the sigmoid function (or logistic function) to transform the linear predictions into probabilities. I implemented both a binomial logistic regression model and multinomial logistic regression models for binary classification, using the lm() and multinom() functions from the nnet library. My main intention was to explore different R functions to see if they performed similarly. Logistic regression is very similar to the LDA algorithm and often they result in similar results, however, generally, Logistic Regression is considered safer and more robust than LDA.

Logistic regression had the lowest testing error (0.127), followed by LDA (0.239). These two models performed similarly, which confirms LDA and logistic regression are very similar models and perform similarly, however logistic regression is generally considered safer and also had very similar training errors.

K-Nearest Neighbors (KNN) model is an algorithm that classifies observations based on their nearest neighbors. I tested multiple values of k (1, 3, 5, 7, 9, 15) to determine the optimal choice. I used the knn() function from the class library in R to implement the models.
K=1 performed the best out of the other k-values. It had the lowest testing error compared to the other k-values at (0.126), which is actually equivalent to the LDA model testing error which is the 2nd overall best model (2nd lowest testing error) in terms of testing error.Higher values of k resulted in lower testing error.

PCA-KNN was the last model implemented. Principal Component Analysis (PCA) was used to reduce dimensionality before applying KNN. High dimensional data can cause issues, so PCA extracts the most informative features before classification. After transforming the dataset into principal components, I applied KNN for classification.

Results:

Training and testing errors were computed and stored for each model. To ensure model robustness, I performed Monte Carlo cross-validation with 100 iterations and computed the mean test error and variance for each model. The results are provided in the tables below.

Models	Training Error	Testing Error
PCA_KNN_k5		0.1139241
PCA_KNN_k3		0.1139241
Logistic R multi	0.0798722	0.1265823
Logistic R binomial	0.0798722	0.1265823
KNN_k1		0.1265823
PCA_KNN_k7		0.1265823
LDA	0.07028754	0.1392405
KNN_k3		0.1392405
KNN_k7		0.1392405
KNN_k9		0.1392405
KNN_k15		0.1392405
Naive Bayes	0.08306709	0.1518987
KNN_k5		0.1518987
PCA_KNN_k1		0.1518987
PCA_KNN_k9		0.1518987
PCA_KNN_k15		0.1518987
QDA	0.06709265	0.1772152

Figure 3A.Training and Testing error for one split.

Models	Variance	Mean Test Error
PCA_KNN_k5	0.001671149	0.07538462
PCA_KNN_k3	0.001676927	0.07871795
PCA_KNN_k9	0.00200951	0.08435897
PCA_KNN_k15	0.002036074	0.08435897
LDA	0.001957444	0.08589744
PCA_KNN_k7	0.002052345	0.08615385
PCA_KNN_k1	0.001843285	0.08666667
QDA	0.001890303	0.08820513
Logistic Regression	0.001493967	0.08923077
Logistic R Binomial	0.001493967	0.08923077
Naive Bayes	0.00228312	0.0925641
KNN_k5	0.002198182	0.12051282
KNN_k3	0.002115766	0.12076923
KNN_k9	0.002445494	0.12205128
KNN_k7	0.002149769	0.12230769
KNN_k1	0.002847343	0.12692308
KNN_k15	0.002736105	0.12820513

3B. Mean Test error and Variance using cross-validation B=100

Model Performance Without Cross-Validation

The initial models without cross validation, showed the PCA_KNN model with k=3 or 5, achieved the lowest test errors. Logistic regression followed closely behind. The worst performing model from the initial models was the QDA model, which had the highest testing error of 0.177.

Model Performance With Monte Carlo Cross-Validation

Monte Carlo cross-validation confirmed that the PCA models combined with KNN (PCA-KNN) consistently had the lowest testing error. Specifically, the PCA_KNN model with k=5 performed the best, followed by k=3, k=9, and then k=15. These models also showed very low variance.

LDA performed the next best after the PCA_KNN models with k=5,3,9, and 15. It outperformed the QDA model, suggesting that the assumption of equal covariance matrices among classes was reasonable for this dataset. Logistic regression had the lowest variance (0.0015), making it the most stable model, although it was slightly less accurate than PCA-KNN. Although Logistic regression had the lowest variance, PCA_KNN was the best balance between variance and testing error. It had the lowest testing error and consistently has the next lowest variance. Naïve Bayes underperformed compared to other models, likely due to the independence assumption among features, which may not hold in this dataset.

The KNN models without PCA consistently performed the worst across all k values. The KNN models had the lowest testing error and variance out of all the models. This shows KNN on its own is not a good classification algorithm for our data. This confirms applying PCA before KNN significantly improves performance.

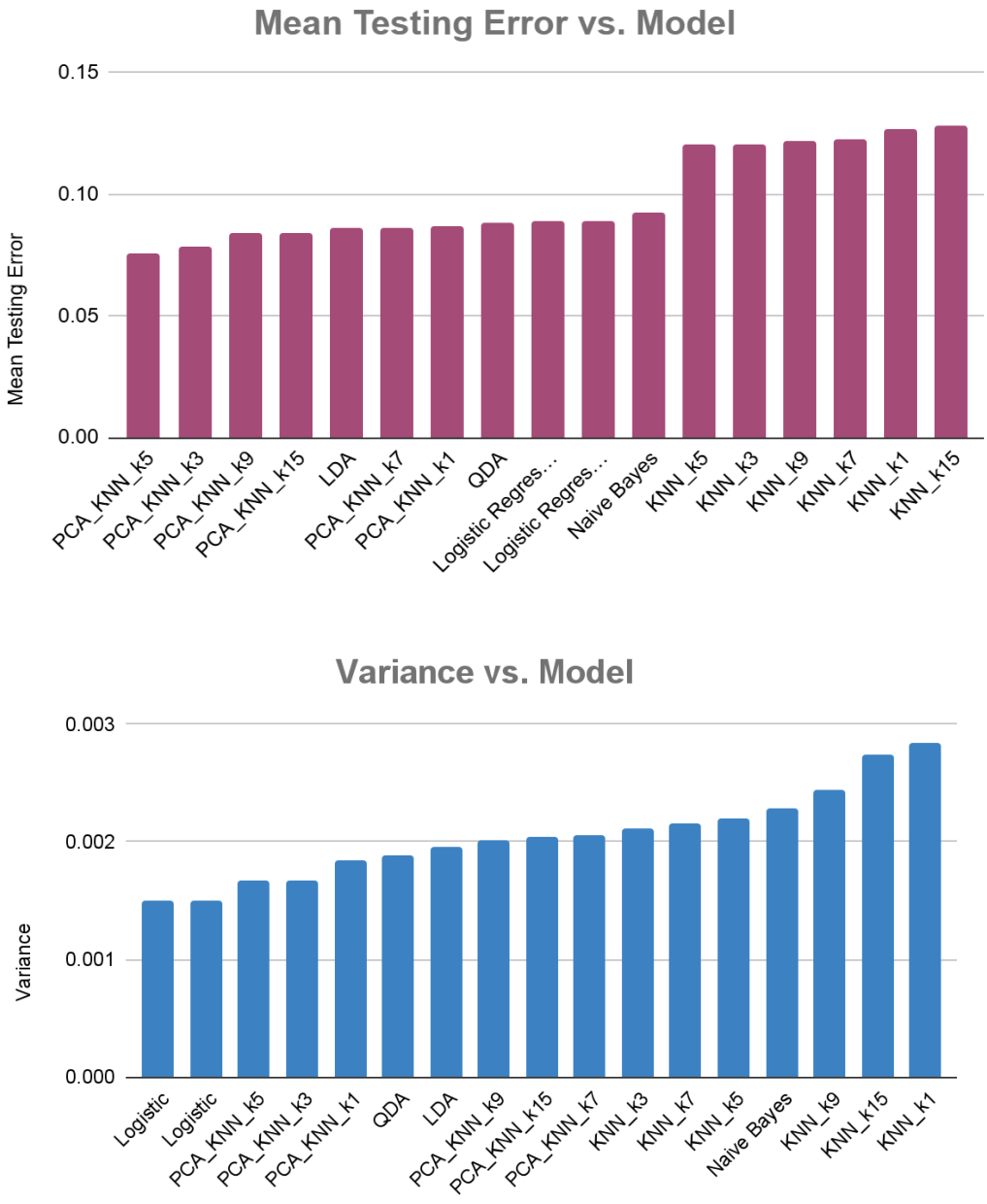

Figure 4. Variance and Mean Testing errors for each mode using Monte-Carlo Cross-Validation with B=100 iterations.

Conclusion:

Overall, this analysis highlights that dimensionality reduction (PCA) can significantly enhance model performance when combined with KNN. The best overall model was the PCA-KNN with k=5, achieving the lowest testing error and maintaining stability. It has the best balance of accuracy and reliability. Logistic

Regression was the most stable model but not as accurate as PCA-KNN. LDA performed well, while QDA and Naïve Bayes struggled.

Car manufacturers can use these insights to focus on features that contribute to high fuel efficiency. Consumers looking for fuel-efficient cars should prioritize attributes like lower weight, fewer cylinders, and higher acceleration, as these are strongly correlated with higher MPG.

Moving forward, future work could explore ensemble methods or deep learning techniques to further refine classification accuracy and enhance model performance for fuel efficiency prediction.

Appendix:

```
#read in data
Auto1 <- read.table(file = "Auto.csv", sep = ",", header=T);

mpg01 = factor(Auto1$mpg >= median(Auto1$mpg), labels = c("Low", "High"))
Auto = data.frame(mpg01, Auto1[,-1]); ##replace "mpg" with "mpg01".
str(Auto)

################################### EDA #####################################

#check for missing values
sum(is.na(Auto))

#check structure & summary
str(Auto)
summary(Auto)

# Distribution of DV
plot(Auto$mpg01, main= 'Distribution of MPG')

# Boxplots: DV vs IV
library(ggplot2)
par(mfrow=c(2,4))
features <- c("cylinders", "displacement", "horsepower", "weight", "acceleration", "year", "origin")

for (feature in features) {
  boxplot(Auto[[feature]] ~ Auto$mpg01,
          main = paste("Boxplot of", feature, "by mpg01"),
          xlab = "MPG Category",
          ylab = feature,
          col = c("lightblue", "lightcoral"))
}

#Scatterplots: linear relationships between variables.
pairs(Auto[, -1], col = ifelse(Auto$mpg01 == TRUE, "blue", "red"))
plot(Auto$horsepower, Auto$weight)

#Correlations
cor_matrix = cor(Auto[-1])

Auto_numeric <- Auto
Auto_numeric$mpg01 <- as.numeric(Auto_numeric$mpg01) #False=1, true=2
cor_matrix <- cor(Auto_numeric)
cor_matrix

library(corrplot)
corrplot(cor_matrix, method = 'color', diag = TRUE,
         col = colorRampPalette(c("navy", "lightblue", "white", "pink", "darkred"))(200),
         tl.col = "black", tl.cex = 0.8,
         number.cex = 0.7,
         addCoef.col = "black")
```

```
#Distribution of IVs
hist(Auto$horsepower)
hist(Auto$weight)

################################### PRE-PROCESSING ###################################
#Variable Selection
#Auto2 = Auto[,c(1:5,8)] #cylinders, displacement, horsepower, weight, &origin are the most important variables

#Split data
n = dim(Auto)[1] ### total number of observations
n1 = round(n/10) ### number of observations randomly selected for testing data

set.seed(19930419); ### set the random seed
flag = sort(sample(1:n, n1))

train_data = Auto[-flag,]
test_data = Auto[flag,]

table(train_data$mpg01)  # Verify class balance in train set
table(test_data$mpg01)

TrainErr <- NULL;
TestErr  <- NULL;

################################### MODELING ###################################

### Method 1: LDA
library(MASS)
#fit1 <- lda(mpg01 ~ ., data= train_data)
mod1 <- lda(train_data[,2:8], train_data[,1]);
#summary(fit1)
summary(mod1)
## training error
pred1 <- predict(mod1, train_data[,2:8])$class;
TrainErr <- c(TrainErr, mean(pred1 != train_data$mpg01));
TrainErr;
## testing error
pred1test <- predict(mod1, test_data[,2:8])$class;
TestErr <- c(TestErr, mean(pred1test != test_data$mpg01));
TestErr;
table(pred1test,  test_data$mpg01)



## Method 2: QDA
mod2 <- qda(train_data[,2:8], train_data[,1])
#mod2 <- qda(mpg01 ~ ., data=train_data)
## Training Error
pred2 <- predict(mod2, train_data[,2:8])$class
TrainErr <- c(TrainErr, mean( pred2!= train_data$mpg01))
TrainErr
## 0.06709265 for miss.class.train.error of QDA 0.07028754, which is much smaller than LDA
##  Testing Error
```

```
TestErr <- c(TestErr, mean(predict(mod2, test_data[,2:8])$class != test_data$mpg01))
TestErr
## Method 3: Naive Bayes
library(e1071)
#mod3 <- naiveBayes(mpg01 ~ ., data = train_data)
mod3 <- naiveBayes(train_data[,2:8], train_data[,1])
## Training Error
pred3 <- predict(mod3, train_data[,2:8]);
TrainErr <- c(TrainErr, mean( pred3 != train_data$mpg01))
TrainErr
## Testing Error
TestErr <- c(TestErr,  mean( predict(mod3,test_data[,2:8]) != test_data$mpg01))
TestErr



### Method 4: (multinomial) logisitic regression)
library(nnet)
mod4 <- multinom(mpg01 ~ ., data=train_data)
summary(mod4);
## Training Error
TrainErr <- c(TrainErr, mean(predict(mod4, train_data[,2:8]) != train_data$mpg01))
TrainErr
## Testing Error of (multinomial) logisitic regression
TestErr <- c(TestErr, mean(predict(mod4, test_data[,2:8]) != test_data$mpg01) )
TestErr


# (binary) Logistic Regression
glm1 <- glm(mpg01 ~ .,  data=train_data, family = binomial)
summary(glm1)  # weight and year are significant with low p-values.
##Train error
log_probs <- predict(glm1, newdata = train_data, type="response");
log_pred <- ifelse(log_probs > 0.5, "High", "Low")  # Map to categorical labels
log_pred <- factor(log_pred, levels = levels(train_data$mpg01)) #threshold
TrainErr <- c(TrainErr, mean(log_pred != train_data$mpg01))
#Test error
log_probs <- predict(glm1, newdata = test_data, type="response");
log_pred <- ifelse(log_probs > 0.5, "High", "Low")  # Map to categorical labels
log_pred <- factor(log_pred, levels = levels(test_data$mpg01)) #threshold
TestErr <- c(TestErr, mean(log_pred != test_data$mpg01))

#plot(Auto$weight, Auto$mpg01)
#lines(Auto$weight, fitted.values(glm1), col="red")

model_error = data.frame('Models' = c("LDA", "QDA", "Naive Bayes", "Logistic R-multinomial", "Logistic
R-binomial"),
                         "Training Error"= TrainErr, "Testing Error"= TestErr)
model_error


## Method 5: KNN
library(class)
```

```
train_x <- train_data[, -1]
test_x <- test_data[, -1]
train_y <- as.numeric(train_data$mpg01) - 1
test_y <- as.numeric(test_data$mpg01) - 1

k_values <- c(1, 3, 5, 7, 9, 15)
knn_results <- data.frame("k" = integer(), "test_error" = numeric())

for (k in k_values) {
  knn_pred <- knn(train_x, test_x, train_y, k = k)

  knn_error <- mean(knn_pred != test_y)
  knn_results <- rbind(knn_results, data.frame("k" = k, "test_error" = knn_error))
}
knn_results


## Method 6: PCA-KNN
pca_model <- prcomp(train_x, scale=TRUE)

# Transform training and test sets using PCA
train_pca <- predict(pca_model, train_x)
test_pca <- predict(pca_model, test_x)

# Use first few principal components (e.g., top 5)
train_pca_reduced <- train_pca[, 1:5]
test_pca_reduced <- test_pca[, 1:5]

# Test KNN with PCA-reduced data
pca_knn_results <- data.frame("k" = integer(), "test_error" = numeric())

for (k in k_values) {
  knn_pca_pred <- knn(train_pca_reduced, test_pca_reduced, train_y, k = k)

  knn_pca_error <- mean(knn_pca_pred != test_y)
  pca_knn_results <- rbind(pca_knn_results, data.frame(k = k, test_error = knn_pca_error))
}
pca_knn_results

################################### EVALUATION ######################################

# Model errors
model_error
knn_results
pca_knn_results

################################### CV ######################################

n = dim(Auto)[1]; # total observations
n1 = round(n/10); # number of observations randomly selected for testing data

set.seed(7406); # Set seed for reproducibility
B= 100; # number of CV loops
```

```
TEALL = NULL; # Final TE values

k_values <- c(1, 3, 5, 7, 9, 15)

for (b in 1:B){
  print(paste("Iteration", b))
  ### randomly select 39 observations as testing data in each loop
  # randomly split into training and testing data
  flag <- sort(sample(1:n, n1));
  train_data <- Auto[-flag,];
  test_data <- Auto[flag,];

  #Prepare Data
  x_train <- as.matrix(train_data[, -1])
  y_train <- train_data$mpg01

  x_test <- as.matrix(test_data[, -1])  # IVs (excluding DV)
  y_test <- test_data$mpg01      # Response


  ### Model 1: LDA
  #fit1 <- lda(mpg01 ~ ., data= train_data)
  mod1 <- lda(train_data[,2:8], train_data[,1]);
  pred1test <- predict(mod1, test_data[,2:8])$class;
  te1 <- mean(pred1test != test_data$mpg01)

  ## Model 2: QDA
  mod2 <- qda(train_data[,2:8], train_data[,1])
  #mod2 <- qda(mpg01 ~ ., data=train_data)
  pred2 <- predict(mod2, train_data[,2:8])$class
  te2 <- mean(predict(mod2, test_data[,2:8])$class != test_data$mpg01)

  ## Model 3: Naive Bayes
  #mod3 <- naiveBayes(mpg01 ~ ., data = train_data)
  mod3 <- naiveBayes(train_data[,2:8], train_data[,1])
  pred3 <- predict(mod3, train_data[,2:8]);
  te3 <- mean(predict(mod3,test_data[,2:8]) != test_data$mpg01)

  ### Model 4: (multinomial) logisitic regression
  mod4 <- multinom(mpg01 ~ ., data=train_data)
  te4 <- mean(predict(mod4, test_data[,2:8]) != test_data$mpg01)

  ### Model 4B: (binomial) logisitic regression
  glm1 <- glm(mpg01 ~ .,  data=train_data, family = binomial)
  log_probs <- predict(glm1, newdata = test_data, type="response");
  log_pred <- ifelse(log_probs > 0.5, "High", "Low")  # Map to categorical labels
  log_pred <- factor(log_pred, levels = levels(test_data$mpg01)) #threshold
  te4B <- mean(log_pred != test_data$mpg01)

  ### Model 5: KNN (for multiple k values)
  knn_errors <- c()
  for (k in k_values) {
    knn_pred <- knn(x_train, x_test, y_train, k = k)
```

```
  knn_error <- mean(knn_pred != y_test)
  knn_errors <- c(knn_errors, knn_error)
 }

 ### Model 6: PCA + KNN
 pca_model <- prcomp(x_train, scale=TRUE)  # PCA on training data
 train_pca <- predict(pca_model, x_train)[, 1:5]  # Keep first 5 PCs
 test_pca <- predict(pca_model, x_test)[, 1:5]

 pca_knn_errors <- c()
 for (k in k_values) {
  pca_knn_pred <- knn(train_pca, test_pca, y_train, k = k)
  pca_knn_error <- mean(pca_knn_pred != y_test)
  pca_knn_errors <- c(pca_knn_errors, pca_knn_error)
 }


 # Store testing errors
 iteration_results <- data.frame(
  Iteration = b,
  "LDA" = te1,
  'QDA' = te2,
  'Naive Bayes' = te3,
  'Logistic Regression' = te4, 'Logistic Regression(binomial)' = te4B,
  'KNN_k1' = knn_errors[1], 'KNN_k3' = knn_errors[2], 'KNN_k5' = knn_errors[3],
  'KNN_k7' = knn_errors[4], 'KNN_k9' = knn_errors[5], 'KNN_k15' = knn_errors[6],
  'PCA_KNN_k1' = pca_knn_errors[1], 'PCA_KNN_k3' = pca_knn_errors[2], 'PCA_KNN_k5' =
pca_knn_errors[3],
  'PCA_KNN_k7' = pca_knn_errors[4], 'PCA_KNN_k9' = pca_knn_errors[5], 'PCA_KNN_k15' =
pca_knn_errors[6]
 )

 TEALL = rbind(TEALL, iteration_results);
}
dim(TEALL); ### Should be B x (number of models)

head(TEALL)

## Sample mean and sample variances for the seven models
mean_errors <- apply(TEALL[, -1], 2, mean);
mean_errors
variance_errors <- apply(TEALL[, -1], 2, var);
Variance_errors

################################### FINAL EVALUATION ######################################
# Model errors
sort_by.data.frame(model_error, model_error$Testing.Error)
sort_by.data.frame(knn_results, knn_results$test_error)
sort_by.data.frame(pca_knn_results, pca_knn_results$test_error)

# Model errors with CV
sort(mean_errors)
sort(variance_errors)
```