Machine Learning Implementations

Tingfang Wang

Table of contents

Introduction

This is a comprehensive collection of machine learning implementations I applied to real-world datasets using both R and Python. It features classification models such as KNN, LDA, and QDA, along with regression techniques like linear, ridge, and lasso logistic regression. Dimensionality reduction methods, including PCR and PCA, and clustering with K-means are also included. Ensemble methods like bagging and boosting enhance predictive accuracy, while LOOCV ensures robust model validation. Additionally, I employed variable selection techniques to identify key predictors, showcasing practical applications of machine learning across diverse scenarios.

Bias-Variance Tradeoff

Consider the following general model for the training data $(Y_i, x_i), i = 1, ..., n$ in a learning problem with quantitative response:

$$Y_i = f(x_i) + \epsilon_i$$

where f is the true mean response function; and the random errors ϵ_i have mean zero, variance σ^2, and are mutually independent. We discussed this model in the class. Let $\hat{f}$ be the estimator of f obtained from the training data. Further, let (x_0, Y_0) be a test observation. In other words, x_0 is a future value of x at which we want to predict Y and Y_0 is the corresponding true value of Y. The test observation follows the same model as the training data, i.e.,

$$Y_0 = f(x_0) + \epsilon_0$$

where ϵ_0 has the same distribution as the ϵ_i for the training data but ϵ_0 is independent of the ϵ_i. Let $\widehat{Y}_0 = \hat{f}(x_0)$ be the predicted value of Y_0.

MSE

Show that $\text{MSE}\{\hat{f}(x_0)\} = (\text{Bias}\{\hat{f}(x_0)\})^2 + \text{var}\{\hat{f}(x_0)\}$

$$
\begin{aligned}
\mathrm{MSE}\{\hat{f}(x_0)\} &= \mathrm{E}\{(\hat{f}(x_0) - f(x_0))^2\} \\
&= \mathrm{E}\{(\hat{f}(x_0) - \mathrm{E}\{\hat{f}(x_0)\} + \mathrm{E}\{\hat{f}(x_0)\} - f(x_0))^2\} \\
&= \mathrm{E}\{(\mathrm{E}\{\hat{f}(x_0)\} - f(x_0))^2 + (\hat{f}(x_0) - \mathrm{E}\{\hat{f}(x_0)\})^2 + 2(\mathrm{E}\{\hat{f}(x_0)\} - f(x_0))(\hat{f}(x_0) - \mathrm{E}\{\hat{f}(x_0)\})\} \\
&= \mathrm{E}\{(\mathrm{E}\{\hat{f}(x_0)\} - f(x_0))^2\} + \mathrm{E}\{(\hat{f}(x_0) - \mathrm{E}\{\hat{f}(x_0)\})^2\} + 2(\mathrm{E}\{\hat{f}(x_0)\} - f(x_0))\mathrm{E}\{(\hat{f}(x_0) - \mathrm{E}\{\hat{f}(x_0)\} \\
&= \mathrm{E}\{(\mathrm{E}\{\hat{f}(x_0)\} - f(x_0))^2\} + \mathrm{E}\{(\hat{f}(x_0) - \mathrm{E}\{\hat{f}(x_0)\})^2\} + 2(\mathrm{E}\{\hat{f}(x_0)\} - f(x_0))(\mathrm{E}\{\hat{f}(x_0)\} - \mathrm{E}\{\hat{f}(x_0) \\
&= \mathrm{E}\{(\mathrm{E}\{\hat{f}(x_0)\} - f(x_0))^2\} + \mathrm{E}\{(\hat{f}(x_0) - \mathrm{E}\{\hat{f}(x_0)\})^2\} + 0 \\
&= (\mathrm{Bias}\{\hat{f}(x_0)\})^2 + \mathrm{var}\{\hat{f}(x_0)\}
\end{aligned}
$$

Expectation of bias-squared

Show that $\mathrm{E}(\hat{Y}_0 - Y_0)^2 = (\mathrm{Bias}\{\hat{f}(x_0)\})^2 + \mathrm{var}\{\hat{f}(x_0)\} + \sigma^2$

$$
\begin{aligned}
\mathrm{E}\{(\hat{Y}_0 - Y_0)^2\} &= \mathrm{E}\{(\hat{f}(x_0) - f(x_0) - \epsilon_0)^2\} \\
&= \mathrm{E}\{(\hat{f}(x_0) - f(x_0))^2 + \epsilon_0^2 - 2\epsilon_0(\hat{f}(x_0) - f(x_0))\} \\
&= \mathrm{E}\{(\hat{f}(x_0) - f(x_0))^2\} + \mathrm{E}\{\epsilon_0^2\} - 2\mathrm{E}\{\epsilon_0(\hat{f}(x_0) - f(x_0))\} \\
&= \mathrm{MSE}\{\hat{f}(x_0)\} + (\mathrm{E}\{\epsilon_0\})^2 + \mathrm{var}\{\epsilon_0\} - 2\mathrm{E}\{\epsilon_0\hat{f}(x_0)\} + 2\mathrm{E}\{\epsilon_0 f(x_0)\} \\
&\text{Since f does not depend on data and } \hat{f}(x_0) \quad \text{and} \quad \epsilon_0 \quad \text{are independent} \\
&= \mathrm{MSE}\{\hat{f}(x_0)\} + \mathrm{var}\{\epsilon_0\} - 2\mathrm{E}\{\epsilon_0\}\mathrm{E}\{\hat{f}(x_0)\} + 2f(x_0)\mathrm{E}\{\epsilon_0\} \\
&= \mathrm{MSE}\{\hat{f}(x_0)\} + \sigma^2 - 0 + 0 \\
&\text{By part (a)} \\
&= (\mathrm{Bias}\{\hat{f}(x_0)\})^2 + \mathrm{var}\{\hat{f}(x_0)\} + \sigma^2
\end{aligned}
$$

KNN

In R

Consider the training and test data in the files 1-tranining-data.csv and 1-test-data.csv, respectively, for a classification problem with two classes.

```
# Read the training data
train.data <- read.csv("1-training_data.csv")
head(train.data)
```

```
        x.1        x.2   y
1 1.4439388  0.1314175 yes
2 0.3726628 -0.8782508 yes
3 1.9018229 -2.5146419 yes
4 1.0193237 -0.2262318 yes
5 1.9805583 -2.8940830 yes
6 1.8975980 -0.1363249 yes
```

```
# Read the test data
test.data <- read.csv("1-test_data.csv")
head(test.data)
```

```
        x.1        x.2   y
1 0.8019971 -0.5181012 yes
2 1.9183688 -2.9347913 yes
3 0.8664813 -0.4837162 yes
4 0.6466528 -0.8791003 yes
5 1.9313624  0.4479239 yes
6 2.2826686 -2.1799404 yes
```

Fit KNN models

Fit KNN with K = 1, 2, . . . , 30, 35, . . . , 200.

```
set.seed(3284)
# Define a function to calculate the error rates
get_knn_errors <- function(train.data, test.data, k){
  # Split the training data into two parts
  train.data.x <- train.data %>% dplyr::select(-y)
  train.label <- train.data[,"y"]
```

```r
  # Perform KNN and predict the labels on the training data
  knn.pred.train <- knn(train.data.x, train.data.x, train.label , k = k)
  # The training error rate
  train.error <- mean(knn.pred.train != train.label)
  # Split the test data into two parts
  test.data.x <- test.data %>% dplyr::select(-y)
  test.label <- test.data[,"y"]
  # Perform KNN and predict the labels on the test data
  knn.pred.test <- knn(train.data.x, test.data.x, train.label, k = k)
  # The test error rate
  test.error <- mean(knn.pred.test != test.label)
  return.df <- data.frame(k, train.error, test.error)
  return(return.df)
}

# Define the main function to perform KNN with different K values
get_knn_errors_main <- function(train.data, test.data){
  k.values <- c(seq(1,200,1))
  length.k <- length(k.values)
  errors.df <- data.frame(matrix(NA, nrow = length.k, ncol = 3))
  colnames(errors.df) <- c("K", "train.error", "test.error")
  for (i in 1:length.k){
    k = k.values[i]
    result.k <- get_knn_errors(train.data, test.data, k = k)
    errors.df[i,] <- result.k
  }
  return(errors.df)
}
errors.df <- get_knn_errors_main(train.data, test.data)
head(errors.df)
```

```
  K train.error test.error
1 1      0.0000     0.2280
2 2      0.1070     0.2225
3 3      0.1130     0.1960
4 4      0.1290     0.1965
5 5      0.1340     0.1860
6 6      0.1415     0.1945
```

The output shows here is the training and test error rate when $k \leq 6$. The complete data frame with different value of k from 1 to 200.

Error rates plot

Plot training and test error rates against K.

```
# Make the plot
errors.df %>%
  gather(key = 'ErrorType',
         value = 'ErrorRate',
         train.error,
         test.error) %>%
  ggplot(aes(x = K, y = ErrorRate)) +
  geom_point(aes(color = ErrorType)) +
  geom_line(aes(color = ErrorType, linetype = ErrorType)) +
  labs(title = "The Error Rates Plot") +
  ylab("Error Rate")
```

The training error is an increasing function of k, with the training error being 0 when $k = 1$. The test error rate first decrease with the increase of k, and then stays around 0.17.

The training error rate is what I would expected as it is a decreasing function of the level of flexibility $1/k$. The test error rate should have a inverted U shape when we plot the error against K, however, the shape is not that obvious in this case.

Optimal value of K

```
# Get the k values that minimize the test error rate
(optimal.k <- errors.df %>% filter(test.error == min(test.error)))
```

```
   K train.error test.error
1 80       0.159      0.167
```

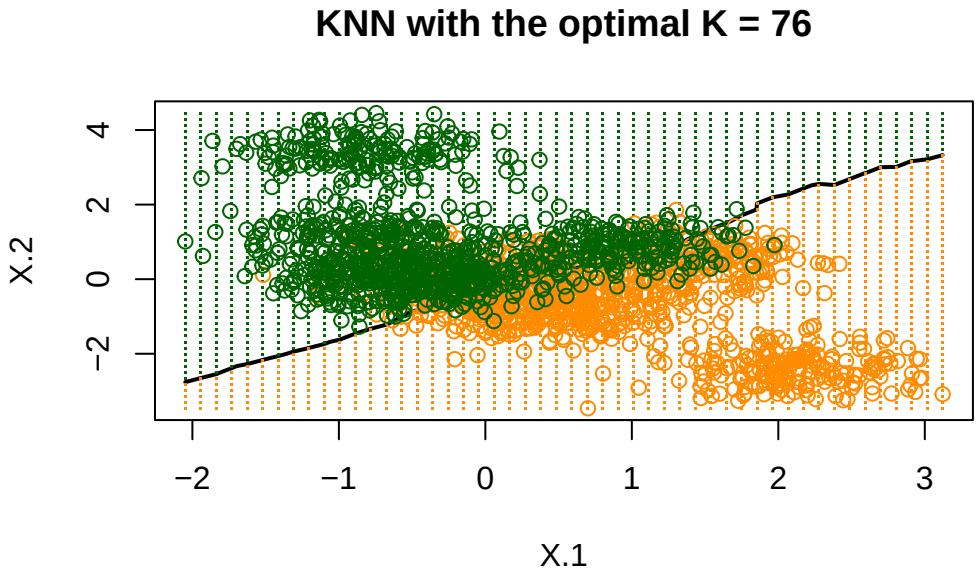
The optimal value of k should be the one that minimizes the test error rate. Thus, the optimal value of k is 80, and the associated training and test error rates are 0.159 and 0.167 respectively.

Decision boundary plot

Plot of the training data that shows the decision boundary for the optimal K.

```
train.data.x <- train.data %>% dplyr::select(-y)
train.label <- train.data[,"y"]

# Create grids for the plot based on the predictors values
x.axis.values <- seq(from = min(train.data.x$x.1), to = max(train.data.x$x.1), length.out = 5
y.axis.values <- seq(from = min(train.data.x$x.2), to = max(train.data.x$x.2), length.out = 5
grid <- expand.grid(x = x.axis.values, y = y.axis.values)

# Fit the model with the optimal value of k = 76
set.seed(1)
knn.model <- knn(train.data.x, grid, train.label, k = 76, prob = TRUE)

# Modified the predicted risk based on the labels
prob <- attr(knn.model, "prob")
prob <- ifelse(knn.model == "yes", prob, 1 - prob)
prob.M <- matrix(prob, 50, 50)

# Create the decision boundary
contour(x.axis.values, y.axis.values,
        prob.M, levels = 0.5,
        xlab = "X.1", ylab = "X.2",
        col = "black", lwd = 2,
        main = "KNN with the optimal K = 76")
points(grid, pch=".",
       col = ifelse(prob.M >= 0.5, "darkorange", "darkgreen"))
```

```r
# Draw the training data points on the plot
points(train.data.x,
       col = ifelse(train.label=="yes", "darkorange", "darkgreen"))
```



The decision boundary is shown as the black line. While there are some overlaps in the training data, the plot demonstrates clear distinctions between the two different labels. This indicates the KNN model's success in correctly classifying most data points within the training dataset.

In Python

Error rates plot

```python
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error

# Set seed for reproducibility
import random
random.seed(3284)
```

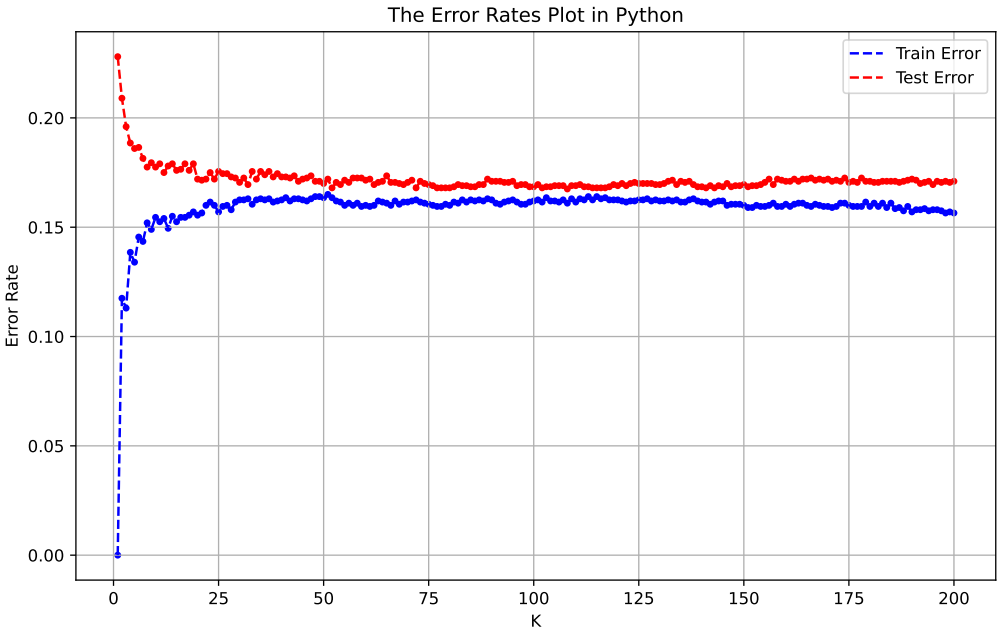
```
# Read the training data
train_data = pd.read_csv("1-training_data.csv")

# Read the test data
test_data = pd.read_csv("1-test_data.csv")

# Define a function to calculate the error rates
def get_knn_errors(train_data, test_data, k):
    train_data_x = train_data.drop(columns=['y'])
    train_label = train_data['y']

    # Perform KNN and predict the labels on the training data
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(train_data_x, train_label)
    knn_pred_train = knn.predict(train_data_x)

    # The training error rate
    train_error = np.mean(train_label != knn_pred_train)

    test_data_x = test_data.drop(columns=['y'])
    test_label = test_data['y']

    # Predict the labels on the test data
    knn_pred_test = knn.predict(test_data_x)

    # The test error rate
    test_error = np.mean(test_label != knn_pred_test)

    return pd.DataFrame({'K': [k],
                         'train.error': [train_error],
                         'test.error': [test_error]})
# Define the main function to perform KNN with different K values
def get_knn_errors_main(train_data, test_data):
    k_values = list(range(1, 201))
    errors_df = pd.DataFrame(columns=["K", "train.error", "test.error"])

    for k in k_values:
        result_k = get_knn_errors(train_data, test_data, k=k)
        errors_df = pd.concat([errors_df, result_k])

    return errors_df
```

```
errors_df = get_knn_errors_main(train_data, test_data)
errors_df.head(6)
```

```
   K  train.error  test.error
0  1       0.0000      0.2280
0  2       0.1175      0.2090
0  3       0.1130      0.1960
0  4       0.1385      0.1885
0  5       0.1340      0.1860
0  6       0.1455      0.1865
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Melt the DataFrame
melted_df = pd.melt(errors_df, id_vars=['K'],
                    value_vars=['train.error', 'test.error'],
                    var_name='ErrorType', value_name='ErrorRate')

# Set up the figure and axis
plt.figure(figsize=(10, 6))

# Create a scatter plot with color differentiation
plt.scatter(melted_df[melted_df['ErrorType'] == 'train.error']['K'],
            melted_df[melted_df['ErrorType'] == 'train.error']['ErrorRate'],
            color='blue', s=10)
plt.scatter(melted_df[melted_df['ErrorType'] == 'test.error']['K'],
            melted_df[melted_df['ErrorType'] == 'test.error']['ErrorRate'],
            color='red', s=10)

# Draw lines for train and test error rates
plt.plot(melted_df[melted_df['ErrorType'] == 'train.error']['K'],
         melted_df[melted_df['ErrorType'] == 'train.error']['ErrorRate'],
         color='blue', linestyle='--', label='Train Error')

plt.plot(melted_df[melted_df['ErrorType'] == 'test.error']['K'],
         melted_df[melted_df['ErrorType'] == 'test.error']['ErrorRate'],
         color='red', linestyle='--', label='Test Error')

# Set titles and labels
plt.title('The Error Rates Plot in Python')
```

```
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```




```
# Close the plot
plt.close()
```

The training and test error rates in Python and R are almost the same, resulting in nearly identical error rate plots.

Optimal k

```
# Find the row(s) with the minimum test error
optimal_k = errors_df[errors_df['test.error'] == errors_df['test.error'].min()]
```

```
# Print the optimal K value(s)
print(optimal_k)
```

```
     K  train.error  test.error
0  108        0.161      0.1675
```

The optimal value of k in Python is 108, and the corresponding training and test error rates are 0.161 and 0.1675, respectively.

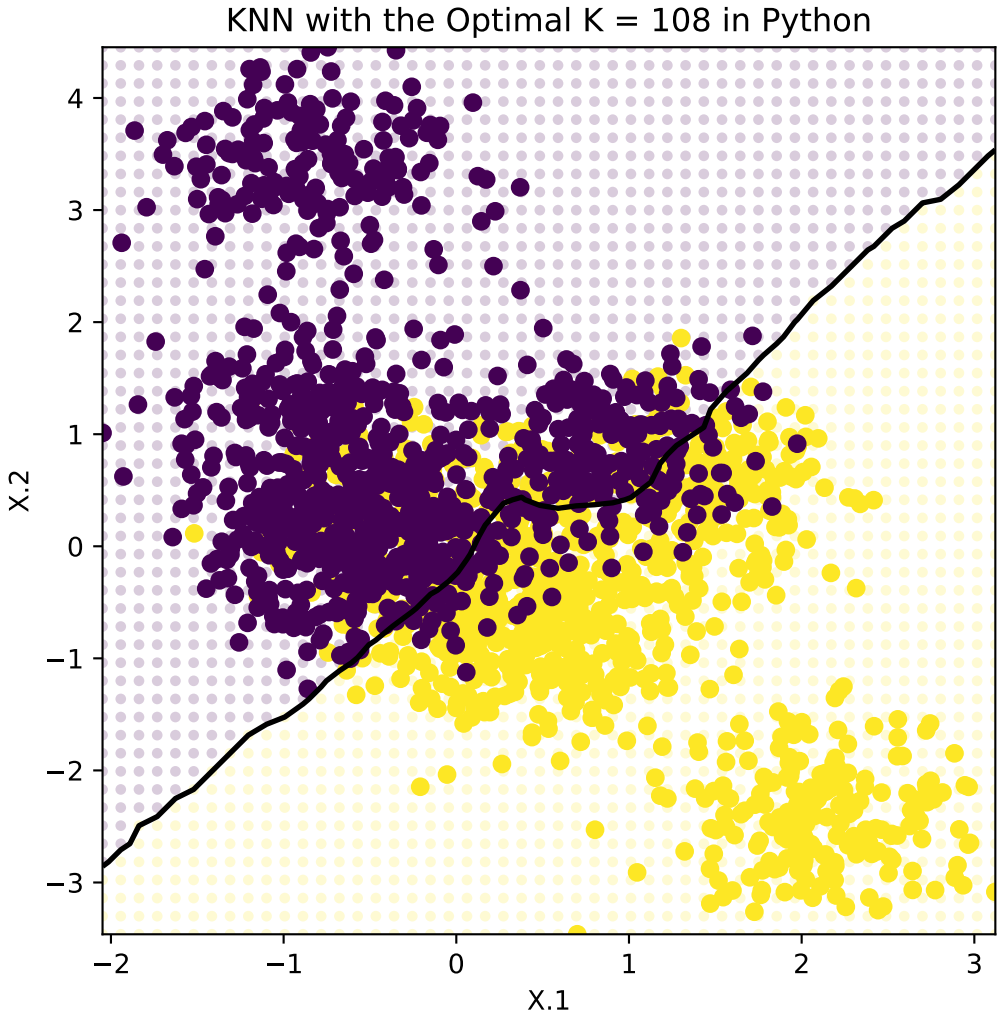
Decision boundary plot

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

# Prepare the data
train_data = pd.read_csv("1-training_data.csv")
train_data_x = train_data.drop(columns=['y'])
train_label = train_data['y']

# Create grids for the plot based on the predictor values
x_axis_values = np.linspace(min(train_data_x['x.1']), max(train_data_x['x.1']), 50)
y_axis_values = np.linspace(min(train_data_x['x.2']), max(train_data_x['x.2']), 50)
x_grid, y_grid = np.meshgrid(x_axis_values, y_axis_values)
grid = pd.DataFrame({'x.1': x_grid.ravel(),
                     'x.2': y_grid.ravel()})

# Fit the model with the optimal value of k = 108
np.random.seed(1)
knn_model = KNeighborsClassifier(n_neighbors=108)
knn_model.fit(train_data_x, train_label)
```

```
KNeighborsClassifier(n_neighbors=108)
```

```
# Predict probabilities
prob = knn_model.predict_proba(grid)[:,1]
prob_M = prob.reshape(50, 50)

# Set up the figure and axis
```

```
plt.figure(figsize=(6, 6))

# Create the decision boundary
plt.contour(x_axis_values, y_axis_values, prob_M, levels=[0.5], colors='black', linewidths=2)
plt.xlabel('X.1')
plt.ylabel('X.2')
plt.title('KNN with the Optimal K = 108 in Python')

# Plot points on the decision boundary
plt.scatter(grid['x.1'], grid['x.2'],
            c=(prob_M >= 0.5).astype(int),
            cmap='viridis', marker='.', alpha=0.2)

# Draw the training data points on the plot
plt.scatter(train_data_x['x.1'],
            train_data_x['x.2'],
            c=(train_label == "yes").astype(int),
            cmap='viridis')

# Show the plot
plt.show()
```




```
# Close the plot
plt.close()
```

Even though the optimal values of k are different, the decision boundary plots are very similar. Both plots clearly demonstrate distinctions between the two different labels in the training data.

Multiple Linear Regression

The prostate cancer dataset consists of data on 97 men with advanced prostate cancer. We would like to understand how PSA level is related to the other predictors in the dataset.

```
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")
head(prostate_cancer)
```

```
  subject   psa cancervol weight age benpros vesinv capspen gleason
1       1 0.651    0.5599 15.959  50       0      0       0       6
2       2 0.852    0.3716 27.660  58       0      0       0       7
3       3 0.852    0.6005 14.732  74       0      0       0       7
4       4 0.852    0.3012 26.576  58       0      0       0       6
5       5 1.448    2.1170 30.877  62       0      0       0       6
6       6 2.160    0.3499 25.280  50       0      0       0       6
```

In R

Correlation check

Since we are interested in understand how PSA level is related to the other predictors, let's first check the correlation of PSA level with the other quantitative predictors.

```
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

# Calculate the correlation matrix between quantitative predictors
variables <- c("psa", "cancervol", "weight",
               "age", "benpros", "capspen",
               "gleason")
cor_matrix <- cor(prostate_cancer[variables])

# Extract the correlation of PSA with other quantitative predictors
psa_correlations <- matrix(cor_matrix["psa", ],
                           nrow = 1)
colnames(psa_correlations) <- c("psa", "cancervol",
                                "weight", "age",
                                "benpros", "capspen",
                                "gleason")

# Print the correlation as a table
```

```r
kable(psa_correlations,
      caption = "Correlation of PSA Level with Other Quantitative
      Variables")
```

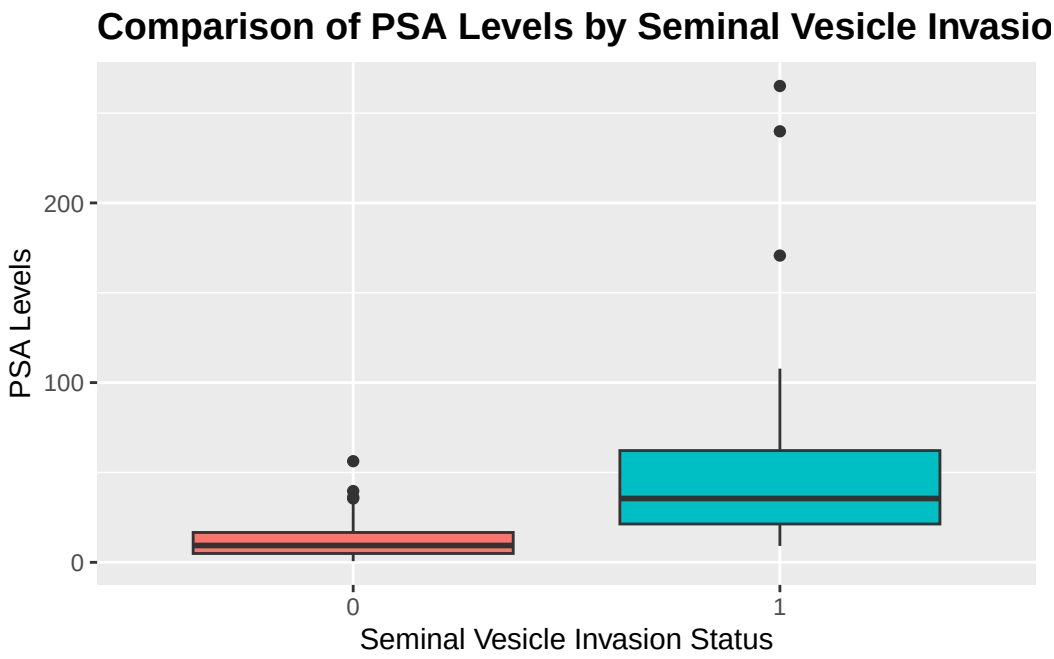
Table 1: Correlation of PSA Level with Other Quantitative Variables

psa	cancervol	weight	age	benpros	capspen	gleason
1	0.6241506	0.0262134	0.0171994	-0.0164865	0.5507925	0.4295798

From the correlation coefficients of PSA level with other quantitative predictors, we see that the strongest positive linear relationships with PSA level are observed for 'cancervol', 'capspen', and 'gleason', indicating that as these factors increases linearly, there is a corresponding tendency for PSA levels to rise linearly as well. The variables 'weight' and 'age' show almost no linear relationship with PSA levels in this dataset. And, the negative linear relationship with 'benpros' is extremely weak.

Since 'vesinv' is a categorical variable, we create a side by side box plot to compare PSA levels across different categories of 'vesinv'.

```r
# Create the side-by-side box plot
prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)
ggplot(prostate_cancer,
       aes(x = vesinv, y = psa, fill = vesinv, group = vesinv)) +
  geom_boxplot() +
  labs(title = "Comparison of PSA Levels by Seminal Vesicle Invasion Status",
       x = "Seminal Vesicle Invasion Status",
       y = "PSA Levels") +
  theme(legend.position = "none",
        axis.title = element_text(size = 11),
        plot.title = element_text(size = 14, face = "bold"))
```



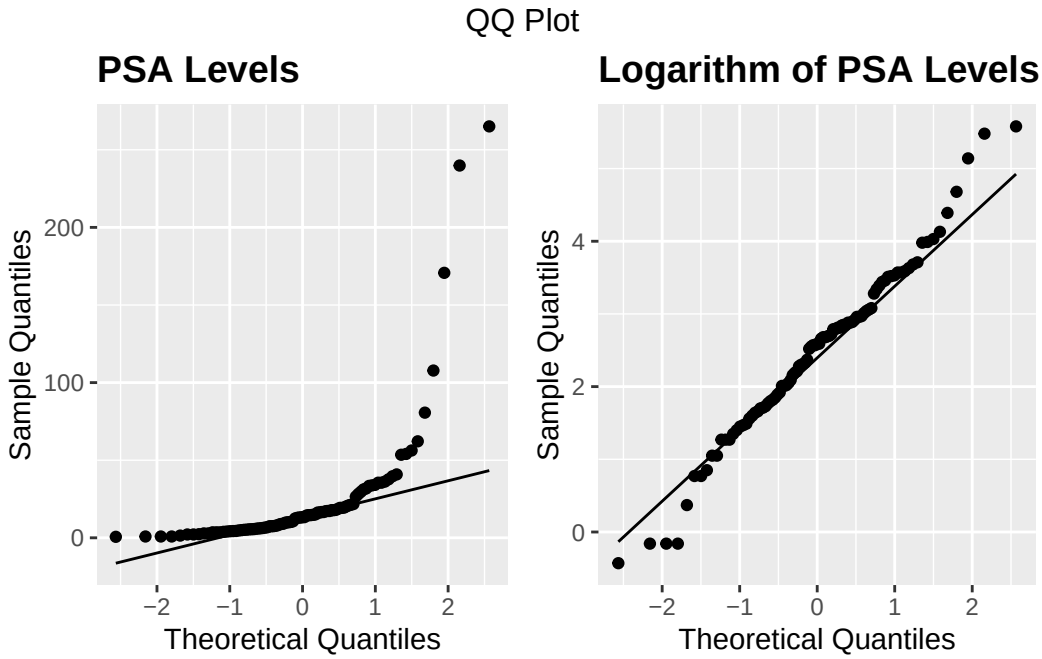
From the box plot, we observe that in cases where Seminal Vesicle Invasion is absent, the distribution of PSA levels tends to display lower quantiles and a narrower interquartile range compared to instances where Seminal Vesicle Invasion is present. This indicates that the absence of Seminal Vesicle Invasion is generally associated with lower PSA levels.

QQ plot

To evaluate if psa is appropriate as a response variable in a linear regression, we check if psa is normally distributed by creating a QQ plot.

```
# Create a QQ plot for PSA levels
qqplot1 <- ggplot(prostate_cancer, aes(sample = psa)) +
  stat_qq() +
  stat_qq_line() +
  labs(title = 'PSA Levels',
       x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  theme(axis.title = element_text(size = 11),
        plot.title = element_text(size = 14, face = "bold"))

# Create a QQ plot for PSA levels
qqplot2 <- ggplot(prostate_cancer,
                  aes(sample = log(psa))) +
```

```
  stat_qq() +
  stat_qq_line() +
  labs(title = "Logarithm of PSA Levels",
       x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  theme(axis.title = element_text(size = 11),
        plot.title = element_text(size = 14, face = "bold"))

grid.arrange(qqplot1 +
               theme(plot.title = element_text(size = 14)),
             qqplot2 +
               theme(plot.title = element_text(size = 14)),
             top = "QQ Plot",
             nrow = 1)
```



We see that the QQ plot of PSA levels is right-skewed, indicating that there are individuals with higher PSA values than what would be expected in a normal distribution. The normality assumption does not hold, a transformation of PSA is necessary, and to shrink the larger values we take the natural log transformation on PSA, and the QQ plot of the logarithm-transformed PSA levels demonstrates a better adherence to the normality assumption.

Hypothesis testing

```r
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

# Create a new column "log_psa" to store the logarithm of the PSA levels
prostate_cancer$log_psa <- log(prostate_cancer$psa)

# Change the integer columns to numeric and vesinv to a factor
prostate_cancer <- prostate_cancer %>%
  mutate(across(where(is.integer), as.numeric))
prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Define a function to fit the simple linear regression models
fit_simple_LR <- function(data = prostate_cancer) {
  # Initialize an empty data frame to store results
  results <- data.frame(Predictor = character(0),
                        P_Value = numeric(0),
                        Adj_R_Square = numeric(0))
  response <- "log_psa"
  predictors <- colnames(data)[3:9]

  # Create a list to store the saved plots
  plots_list <- list()

  # Iterate through each predictor
  for (i in 1:length(predictors)) {
    predictor <- predictors[i]
    # Fit a simple linear regression model
    model <- lm(paste(response, "~", predictor),
                data = data)

    # Get p-value from t-test
    p_value <- summary(model)$coefficients[2, 4]

    # Get adjusted R-squared
    adj_r_square <- summary(model)$adj.r.squared

    # Store results in data frame
    results <- rbind(results,
                     data.frame(Predictor = predictor,
                                P_Value = p_value,
                                Adj_R_Square = adj_r_square))
```

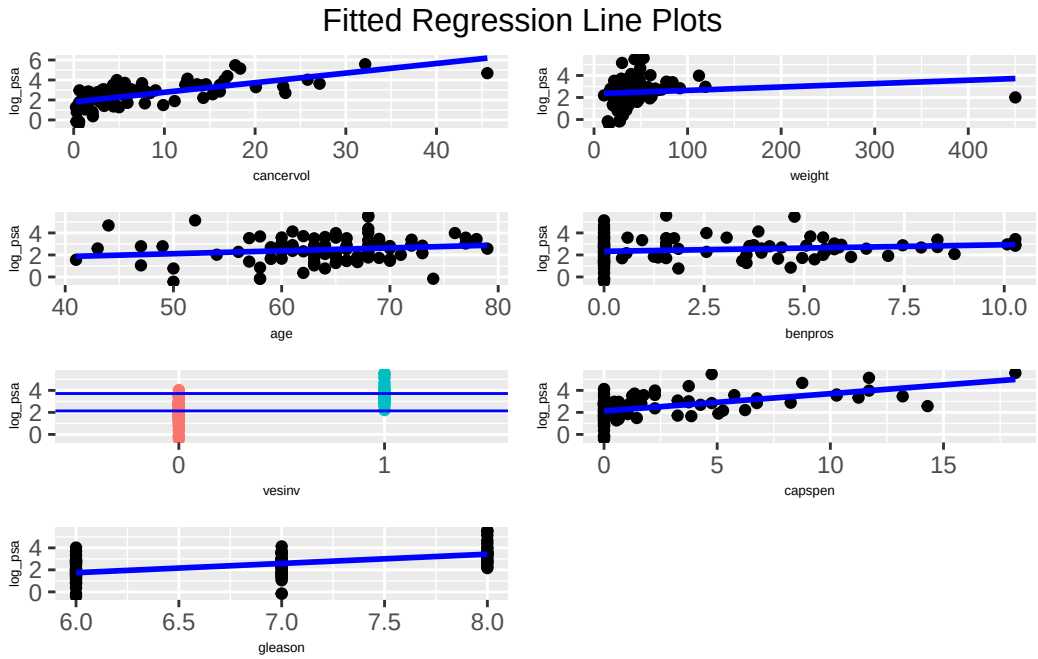
```
    if (is.factor(data[,predictor])) {
      # Create the plot
      plot <- ggplot(data,
                     aes(x = .data[[predictor]],
                         y = .data[[response]],
                         color = .data[[predictor]])) +
        geom_point() +
        geom_hline(yintercept = unique(predict(model)),
                   color = "blue") +
        theme(legend.position = "none") +
        xlab(predictor) + ylab(response)+
        theme(axis.title = element_text(size = 5),
              plot.title = element_text(size = 5,
                                        face = "bold"))
    }
    else {
      # Create plot of the regression line and save it
      plot <- ggplot(data = data,
                     aes(x = .data[[predictor]],
                         .data[[response]])) +
        geom_point() +
        geom_smooth(method = "lm", color = "blue",
                    se = FALSE) +
        xlab(predictor) + ylab(response) +
        theme(axis.title = element_text(size = 5),
              plot.title = element_text(size = 5,
                                        face = "bold"))
    }
    # Add the plot to the list
    plots_list[[i]] <- plot
  }

  return(list("table" = results, "plots" = plots_list))
}

results <- fit_simple_LR(data = prostate_cancer)
# Print the table
kable(results$table,
      caption = "The P-value of the predictor and Adjusted R Square of the models")
```

Table 2: The P-value of the predictor and Adjusted R Square of the models

Predictor	P_Value	Adj_R_Square
cancervol	0.0000000	0.4257645
weight	0.2349550	0.0044456
age	0.0961492	0.0186459
benpros	0.1236227	0.0145097
vesinv	0.0000000	0.3136185
capspen	0.0000001	0.2606463
gleason	0.0000000	0.2830710

Based on the p-values associated with each predictor's t-statistic and the adjusted R-squared in the table. 'cancervol', 'vesinv', 'capspen', and 'gleason' are predictors with statistically significant associations with the response variable, and the corresponding models explain a relatively substantial portion of the variability in the response, as reflected by their relatively high adjusted R-squared values. 'weight', 'age', and 'benpros' do not show strong evidence of significant associations, and they contribute to a relatively smaller proportion of the variability in the response.

```
# Combine the fitted line plots
grid.arrange(grobs = results$plots,
             nrow = 4, ncol = 2,
             top = "Fitted Regression Line Plots")
```



Furthermore, based on the fitted line plots, the lines for 'cancervol', 'vesinv', 'capspen', and 'gleason' closely follow the pattern of the data points. While, the lines for 'weight', 'age', and 'benpros' deviate significantly from the data points.

Model fitting

```
# Fit the model
multi_LR <- lm(log_psa ~ cancervol + weight + age + benpros +
                 vesinv + capspen + gleason,
               data = prostate_cancer)

# Extract coefficients and p-values
coefficients <- summary(multi_LR)$coefficients[, c(1, 4)]

# Create a data frame
coefficients_df <- data.frame(
  Predictor = rownames(coefficients),
  Estimate = coefficients[, 1],
  `Pr(>|t|)` = coefficients[, 2]
)

# Print the coefficients table
```

```
kable(coefficients,
      caption = "Regression Coefficients and P-values")
```

Table 3: Regression Coefficients and P-values

	Estimate	Pr(>|t|)
(Intercept)	-0.6857956	0.4940876
cancervol	0.0694537	0.0000078
weight	0.0013800	0.4507937
age	-0.0027989	0.8118614
benpros	0.0874695	0.0040066
vesinv1	0.7826232	0.0044786
capspen	-0.0265208	0.4217656
gleason	0.3581530	0.0062918

Based on the p-values, we can reject the null hypothesis and conclude that 'cancervol', 'benpros', 'vesinv1', and 'gleason' are statistically significant predictors in this multiple regression model. The other predictors ('weight', 'age', and 'capspen') do not have statistically significant associations with the response variable.

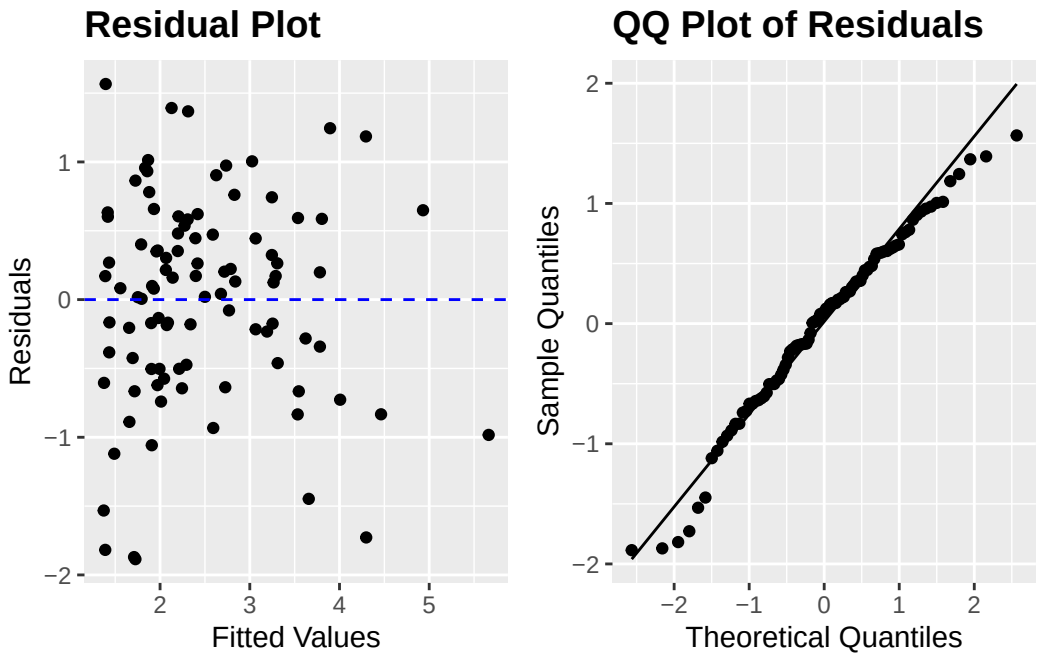
Likelihood ratio test

```
# Fit the model
multi_LR_new <- lm(log_psa ~ cancervol + benpros + vesinv + gleason,
                   data = prostate_cancer)

# Compare this model with the full model
anova(multi_LR_new, multi_LR)
```

```
Analysis of Variance Table

Model 1: log_psa ~ cancervol + benpros + vesinv + gleason
Model 2: log_psa ~ cancervol + weight + age + benpros + vesinv + capspen +
    gleason
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     92 53.229
2     89 52.477  3   0.75232 0.4253 0.7353
```

Based on the p-value(0.7353) in the ANOVA table, it appears that the full model which has all the predictors does not provide a significantly better fit compared to the model that only have the 4 predictors we picked based on part (d).

```
# Make the residual plot
# Create a data frame with fitted values and residuals
residuals <- data.frame(Fitted = fitted(multi_LR_new),
                        Residuals = residuals(multi_LR_new))

# Create the residual plot using ggplot2
residual_plot <- ggplot(residuals,
                        aes(x = Fitted,
                            y = Residuals)) +
  geom_point() +
  geom_hline(yintercept = 0,
             linetype = "dashed",
             color = "blue") +
  labs(title = "Residual Plot",
       x = "Fitted Values",
       y = "Residuals") +
  theme(axis.title = element_text(size = 11),
        plot.title = element_text(size = 14, face = "bold"))

# Create QQ plot
qq_plot <- ggplot(residuals,
                  aes(sample = Residuals)) +
  geom_qq() +
  geom_qq_line() +
  labs(title = "QQ Plot of Residuals",
       x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  theme(axis.title = element_text(size = 11),
        plot.title = element_text(size = 14,
                                  face = "bold"))

grid.arrange(grobs = list(residual_plot, qq_plot),
             nrow = 1, ncol = 2)
```



The residual plot does not have any pattern, suggesting that the relationship between the predictor variables (cancervol, benpros, vesinv, gleason) and the response variable (log_psa) is adequately captured by the linear model. That is, the relationship is reasonably linear. And, the spread of the residuals appears relatively consistent across different levels of the fitted values. This indicates that the assumption of constant variance (homoscedasticity) is likely met.

The absence of apparent skewness in the QQ plot of residuals provides strong evidence that the residuals are approximately normally distributed, indicating that the normality assumption for the residuals is likely being met.

The fitted model

Write the final model in equation form, being careful to handle qualitative predictors (if any) properly.

When vesinv $= 0$

$$\log(\text{psa}) = -0.65013 + 0.06488 \cdot \text{cancervol} + 0.09136 \cdot \text{benpros} + 0.33376 \cdot \text{gleason}$$

And when vesinv $= 1$

$$\log(\text{psa}) = 0.03408 + 0.06488 \cdot \text{cancervol} + 0.09136 \cdot \text{benpros} + 0.33376 \cdot \text{gleason}$$

LDA, QDA, and Naive Bayes

Relation betwwen LDA and Naive Bayes

Consider a classification problem with K classes. The Bayes decision rule assigns an observation to the class for which $P(Y = k \mid x)$, or equivalently, its log odds relative to class K as baseline, i.e., $P(Y = k \mid x) = P(Y = K \mid x)$, is maximum.

Log-odds expression

Assuming that the model parameters are known, verify that the log odds can be expressed as follows.

For **LDA**, we assume $X|Y = k \sim N(\mu_k, \Sigma)$, then

$$\begin{aligned}
\log(\frac{P(Y=k|X=x)}{P(Y=K|X=x)}) &= \log(\frac{\pi_k f_k(x)}{\pi_K f_K(x)}) \\
&= \log(\frac{\pi_k \exp(-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k))}{\pi_K \exp(-\frac{1}{2}(x-\mu_K)^T\Sigma^{-1}(x-\mu_K))}) \\
&= \log(\frac{\pi_k}{\pi_K}) - \frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k) \\
&\quad + \frac{1}{2}(x-\mu_K)^T\Sigma^{-1}(x-\mu_K) \\
&= \log(\frac{\pi_k}{\pi_K}) - \frac{1}{2}(\mu_k+\mu_K)^T\Sigma^{-1}(\mu_k-\mu_K) \\
&\quad + x^T\Sigma^{-1}(\mu_k-\mu_K) \\
&= a_k + \sum_{j=1}^{p} b_{kj}x_j
\end{aligned}$$

where

$$a_k = log(\frac{\pi_k}{\pi_K}) - \frac{1}{2}(\mu_k+\mu_K)^T\Sigma^{-1}(\mu_k-\mu_K)$$

and, b_{kj} is the jth component of $\Sigma^{-1}(\mu_k - \mu_K)$

For **naive Bayes classifier**, we have

$P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$ and, $f_k(x) = f_{k1}(x_1) \times f_{k2}(x_2) \times ... \times f_{kp}(x_p)$

Therefore, we can write

$$\begin{aligned}
\log(\frac{P(Y=k|X=x)}{P(Y=K|X=x)}) &= \log(\frac{\pi_k f_k(x)}{\pi_K f_K(x)}) \\
&= \log(\frac{\pi_k \prod_{j=1}^{p} f_{kj}(x_j)}{\pi_K \prod_{j=1}^{p} f_{Kj}(x_j)}) \\
&= \log(\frac{\pi_k}{\pi_K}) + \sum_{j=1}^{p} \log(\frac{f_{kj}(x_j)}{f_{Kj}(x_j)}) \\
&= a_k + \sum_{j=1}^{p} g_{kj}(x_j)
\end{aligned}$$

where $a_k = \log(\frac{\pi_k}{\pi_K})$ and $g_{kj}(x_j) = \log(\frac{f_{kj}(x_j)}{f_{Kj}(x_j)})$.

Assumptions of LDA to be naive Bayes?

Under the assumption that features are normally distributed with a common within-class covariance matrix, the decision boundary of LDA is when $\pi_k f_k(x) - \pi_K f_K(x) = 0$. That is,

$$(\hat{\mu}_k - \hat{\mu}_K)^T \hat{\Sigma}^{-1} x = \frac{1}{2}(\hat{\mu}_k + \hat{\mu}_K)^T \hat{\Sigma}^{-1}(\hat{\mu}_k - \hat{\mu}_K) + \log(\frac{\hat{\pi}_K}{\hat{\pi}_k})$$

Thus, the decision boundary of LDA is a linear function of x. As any classifier with a linear decision boundary is a special case of naive Bayes, LDA is also a special case of naive Bayes.

Assumptions of naive Bayes to be LDA?

If we model $f_{kj}(x_j)$ in the naive Bayes classifier using a one-dimensional Gaussian distribution $N(\mu_{kj}, \sigma_j^2)$, then

$$
\begin{aligned}
\log(\frac{P(Y=k|X=x)}{P(Y=K|X=x)}) &= \log(\frac{\pi_k f_k(x)}{\pi_K f_K(x)}) \\
&= \log(\frac{\pi_k}{\pi_K}) + \sum_{j=1}^{p} \log(\frac{f_{kj}(x_j)}{f_{Kj}(x_j)}) \\
&= \log(\frac{\pi_k}{\pi_K}) + \sum_{j=1}^{p} \log(\frac{\exp(-\frac{1}{2}(x_j-\mu_{kj})^2/\sigma_j^2)}{\exp(-\frac{1}{2}(x_j-\mu_{Kj})^2/\sigma_j^2)}) \\
&= \log(\frac{\pi_k}{\pi_K}) + \sum_{j=1}^{p} \{-\frac{1}{2}(x_j-\mu_{kj})^2/\sigma_j^2) + \frac{1}{2}(x_j-\mu_{Kj})^2/\sigma_j^2\} \\
&\text{let } \Sigma \text{ be a diagonal matrix with jth diagonal element equal to } \sigma_j^2 \\
&= \log(\frac{\pi_k}{\pi_K}) - \frac{1}{2}(\mu_k+\mu_K)^T\Sigma^{-1}(\mu_k-\mu_K) + x^T\Sigma^{-1}(\mu_k-\mu_K) \\
&= a_k + \sum_{j=1}^{p} \frac{\mu_{kj}-\mu_{Kj}}{\sigma_j^2} x_j
\end{aligned}
$$

That is, naive Bayes is a special case of LDA with Σ being a diagonal matrix with jth diagonal element equal to σ_j.

In R

The admission officer of a business school has used an "index" of undergraduate grade point average (GPA, X1) and graduate management aptitude test (GMAT, X2) scores to help decide which applicants should be admitted to the school's graduate programs. This index is used categorize each applicant into one of three groups | admit (group 1), do not admit (group 2), and borderline (group 3). Standardize the predictors before performing any analysis. We will take the last five observations in each category as test data and the remaining observations as training data.

```
admission_data <- read.csv("admission.csv")

# Standardize GPA (X1)
admission_data$std_GPA <- (admission_data$GPA -
                             mean(admission_data$GPA)) /
  sd(admission_data$GPA)

# Standardize GMAT (X2)
admission_data$std_GMAT <- (admission_data$GMAT -
                              mean(admission_data$GMAT)) /
```

```
  sd(admission_data$GMAT)

# Identify the last five observations in each category for test data
test_data <- admission_data %>%
  group_by(Group) %>%
  slice_tail(n = 5)
test_data <- data.frame(test_data)

# The remaining data will be used for training
training_data <- admission_data %>%
  anti_join(test_data, by = c("GPA", "GMAT", "Group", "std_GPA", "std_GMAT"))
training_data <- data.frame(training_data)
head(training_data)
```

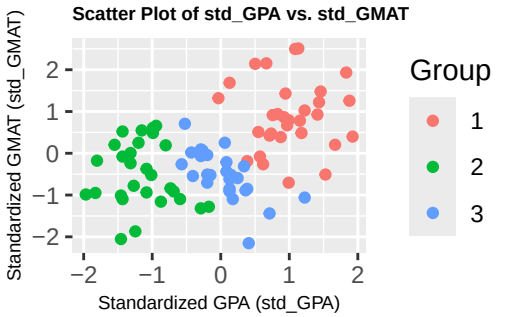
```
   GPA GMAT Group     std_GPA    std_GMAT
1 2.96  596     1 -0.03400558  1.31930625
2 3.14  473     1  0.38557937 -0.18948251
3 3.22  482     1  0.57206156 -0.07908333
4 3.29  527     1  0.73523349  0.47291255
5 3.69  505     1  1.66764447  0.20304790
6 3.46  693     1  1.13150816  2.50916405
```

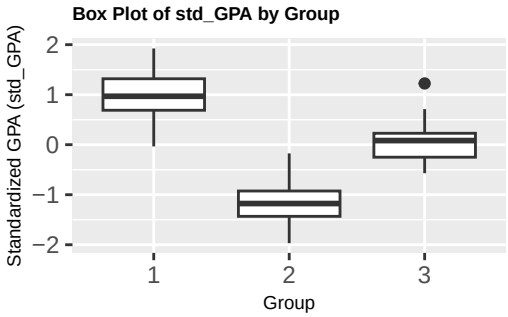
Exploratory analysis

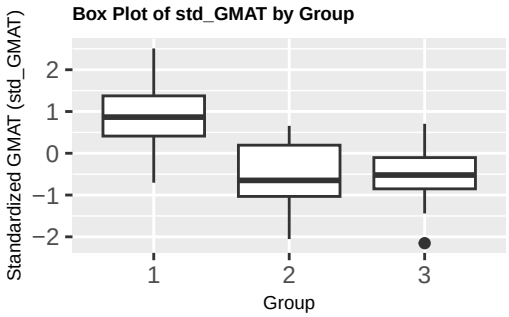
The scatter plot indicates that the groups in the data are well-separated. The box plot of GPA reveals that the 'admit' group exhibits the highest median GPA compared to the other two groups. This suggests that, on average, students in Group 1 tend to have the highest GPAs. The 'do not admit' group demonstrates the lowest median GPA among the three groups. On average, students in Group 2 tend to have lower GPAs compared to the other two groups. The 'borderline' group has a median GPA that falls between Group 1 and Group 2. This implies that, on average, students in Group 3 have GPAs that are higher than those in Group 2 but lower than those in Group 1. The box plot suggests that GPA can be informative in predicting the groups. Similarly, the box plot of GMAT scores displays a similar pattern. While the distinction between groups is not as pronounced as with GPA, it still proves to be useful in predicting the groups.

```
# Create a scatter plot of std_GPA and std_GMAT colored by Group
scatter_plot <- ggplot(data = admission_data,
                       aes(x = std_GPA,
                           y = std_GMAT,
                           color = as.factor(Group))) +
```

```
  geom_point() +
  labs(title = "Scatter Plot of std_GPA vs. std_GMAT",
       x = "Standardized GPA (std_GPA)",
       y = "Standardized GMAT (std_GMAT)",
       color = "Group") +
  theme(axis.title = element_text(size = 7),
        plot.title = element_text(size = 7, face = "bold"))

# Create a box plot of std_GPA by Group
box_1 <- ggplot(data = admission_data,
                aes(x = as.factor(Group),
                    y = std_GPA)) +
  geom_boxplot() +
  labs(title = "Box Plot of std_GPA by Group",
       x = "Group",
       y = "Standardized GPA (std_GPA)") +
  theme(axis.title = element_text(size = 7),
        plot.title = element_text(size = 7, face = "bold"))

# Create a box plot of std_GMAT by Group
box_2 <- ggplot(data = admission_data,
                aes(x = as.factor(Group),
                    y = std_GMAT)) +
  geom_boxplot() +
  labs(title = "Box Plot of std_GMAT by Group",
       x = "Group",
       y = "Standardized GMAT (std_GMAT)")+
  theme(axis.title = element_text(size = 7),
        plot.title = element_text(size = 7, face = "bold"))

grid.arrange(grobs = list(scatter_plot, box_1, box_2),
             nrow = 2, ncol = 2)
```



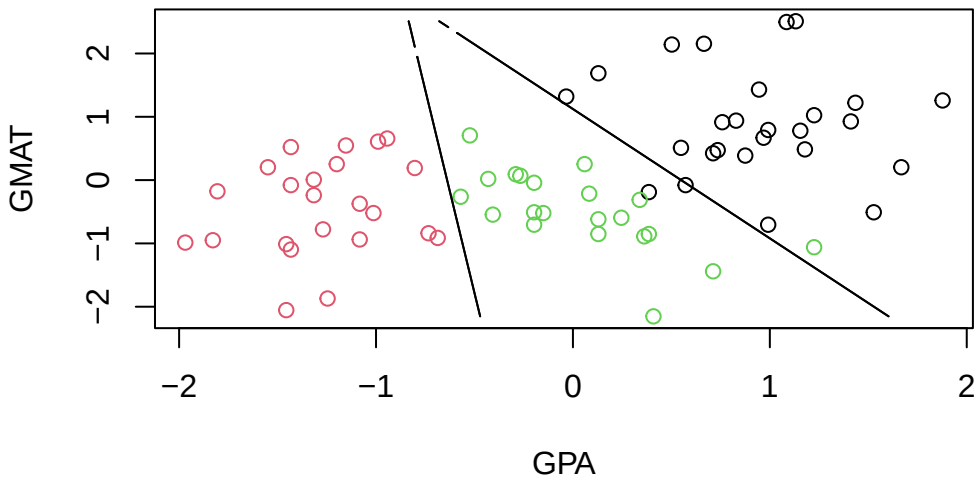




Linear Discriminant Analysis (LDA)

```
training_data <- training_data %>%
  dplyr::select(std_GPA, std_GMAT, Group)
training_data$Group <- as.factor(training_data$Group)
lda.fit <- lda(Group ~ std_GPA + std_GMAT, data = training_data)

# Decision boundary
# Set up a dense grid and compute posterior prob on the grid
n.grid <- 100
x1.grid <- seq(f = min(training_data[, "std_GPA"]),
               t = max(training_data[, "std_GPA"]),
               l = n.grid)
x2.grid <- seq(f = min(training_data[, "std_GMAT"]),
               t = max(training_data[, "std_GMAT"]),
               l = n.grid)
grid <- expand.grid(x1.grid, x2.grid)
colnames(grid) <- c("std_GPA", "std_GMAT")
pred.grid <- predict(lda.fit, grid)

# Define a function to calculate the decision boundary probabilities
subtract_max_other <- function(row) {
```

```
  new_row <- c()
  for (i in 1:3){
    max_other <- max(row[-i])
    new_row[i] <- row[i] - max_other
  }
  return (new_row)
}

# Apply the function row-wise
probs <- t(apply(pred.grid$posterior, 1, subtract_max_other))

# Separate the decision boundary probabilities by Group
prob_1 <- matrix(probs[, 1], nrow = n.grid,
                 ncol = n.grid, byrow = F)
prob_2 <- matrix(probs[, 2], nrow = n.grid,
                 ncol = n.grid, byrow = F)
prob_3 <- matrix(probs[, 3], nrow = n.grid,
                 ncol = n.grid, byrow = F)

# Plot the training data
plot(training_data[, 1:2],
     col = training_data$Group,
     xlab("GPA"), ylab("GMAT"),
     cex = 1, cex.lab = 1)

# Add the decision boundaries to the plot
contour(x1.grid, x2.grid, prob_1,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
contour(x1.grid, x2.grid, prob_2,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
contour(x1.grid, x2.grid, prob_3,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
```




```
# Compute confusion matrix and misclassification rate for training data
train_pred <- predict(lda.fit, training_data)$class
train_confusion <- table(Predicted = train_pred,
                         Actual = training_data$Group)
train_error_rate <- 1 - sum(diag(train_confusion)) /
  sum(train_confusion)

# Compute confusion matrix and misclassification rate for test data
test_pred <- predict(lda.fit, test_data)$class
test_confusion <- table(Predicted = test_pred,
                        Actual = test_data$Group)
test_error_rate <- 1 - sum(diag(test_confusion)) /
  sum(test_confusion)

cat("Confusion Matrix (Training Data | Test Data):\n")
```

```
Confusion Matrix (Training Data | Test Data):
```

```
print(cbind(train_confusion,test_confusion))
```

```
   1  2  3 1 2 3
```

```
1 24  0  1 4 0 0
2  0 23  0 0 3 0
3  2  0 20 1 2 5
```

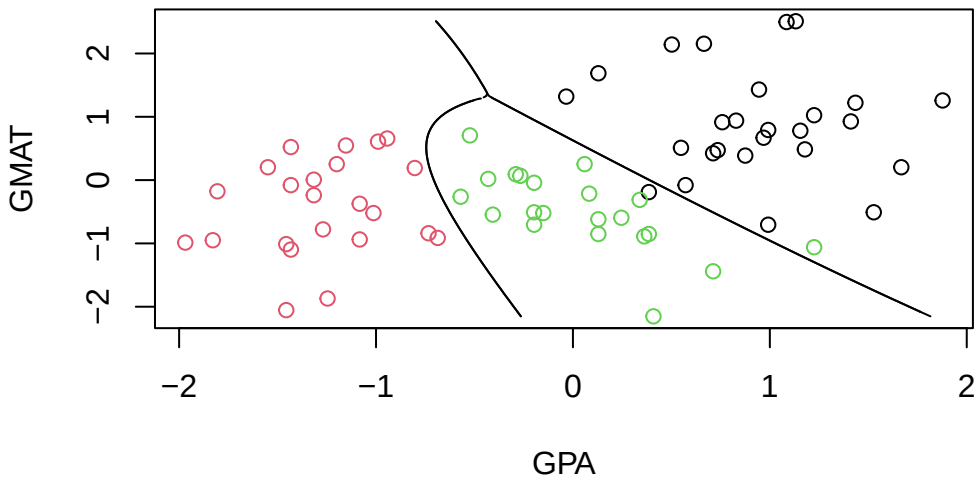
The plot demonstrates clear distinctions between the three different labels. The model correctly classifying all the data points except for only two observations within the training dataset.

The confusion matrix for the training data shows that most of the predictions are accurate, only Class 1 has 2 misclassifications. The misclassification rate is very low (4.29%), indicating that the model performs well on the training data. In the test data, there are more misclassifications compared to the training data. The misclassification rate is higher (20%) compared to the training data. This suggests that the model might have a problem of overfitting.

Quadratic Discriminant Analysis (QDA)

```r
qda.fit <- qda(Group ~ std_GPA + std_GMAT,
               data = training_data)

# Decision boundary
# Set up a dense grid and compute posterior prob on the grid
n.grid <- 100
x1.grid <- seq(f = min(training_data[, "std_GPA"]),
               t = max(training_data[, "std_GPA"]),
               l = n.grid)
x2.grid <- seq(f = min(training_data[, "std_GMAT"]),
               t = max(training_data[, "std_GMAT"]),
               l = n.grid)
grid <- expand.grid(x1.grid, x2.grid)
colnames(grid) <- c("std_GPA", "std_GMAT")
pred.grid <- predict(qda.fit, grid)

# Define a function to calculate the decision boundary probabilities
subtract_max_other <- function(row) {
  new_row <- c()
  for (i in 1:3){
    max_other <- max(row[-i])
    new_row[i] <- row[i] - max_other
  }
  return (new_row)
}
```

```r
# Apply the function row-wise
probs <- t(apply(pred.grid$posterior, 1, subtract_max_other))

# Separate the decision boundary probabilities by Group
prob_1 <- matrix(probs[, 1], nrow = n.grid,
                 ncol = n.grid, byrow = F)
prob_2 <- matrix(probs[, 2], nrow = n.grid,
                 ncol = n.grid, byrow = F)
prob_3 <- matrix(probs[, 3], nrow = n.grid,
                 ncol = n.grid, byrow = F)

# Plot the training data
plot(training_data[, 1:2],
     col = training_data$Group,
     xlab("GPA"), ylab("GMAT"),
     cex = 1, cex.lab = 1)

# Add the decision boundaries to the plot
contour(x1.grid, x2.grid, prob_1,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
contour(x1.grid, x2.grid, prob_2,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
contour(x1.grid, x2.grid, prob_3,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
```




```
# Compute confusion matrix and misclassification rate for training data
train_pred <- predict(qda.fit, training_data)$class
train_confusion <- table(Predicted = train_pred,
                         Actual = training_data$Group)
train_error_rate <- 1 - sum(diag(train_confusion)) /
  sum(train_confusion)

# Compute confusion matrix and misclassification rate for test data
test_pred <- predict(qda.fit, test_data)$class
test_confusion <- table(Predicted = test_pred,
                        Actual = test_data$Group)
test_error_rate <- 1 - sum(diag(test_confusion)) /
  sum(test_confusion)

cat("Confusion Matrix (Training Data | Test Data):\n")
```

```
Confusion Matrix (Training Data | Test Data):
```

```
print(cbind(train_confusion,test_confusion))
```

```
   1  2  3 1 2 3
```

```
1 25  0  1 5 0 0
2  0 23  0 0 3 0
3  1  0 20 0 2 5
```

The plot demonstrates clear distinctions between the three different labels. The model correctly classifying all the data points except for only two observations within the training dataset.

The confusion matrix for the training data shows very good performance. There is only two misclassification in total. The misclassification rate is quite low (2.86%), indicating that the model performs good on the training data. In the test data, there are also two misclassifications. The misclassification rate is higher (13.33%) compared to the training data, but it's still relatively low compare to the LDA model.

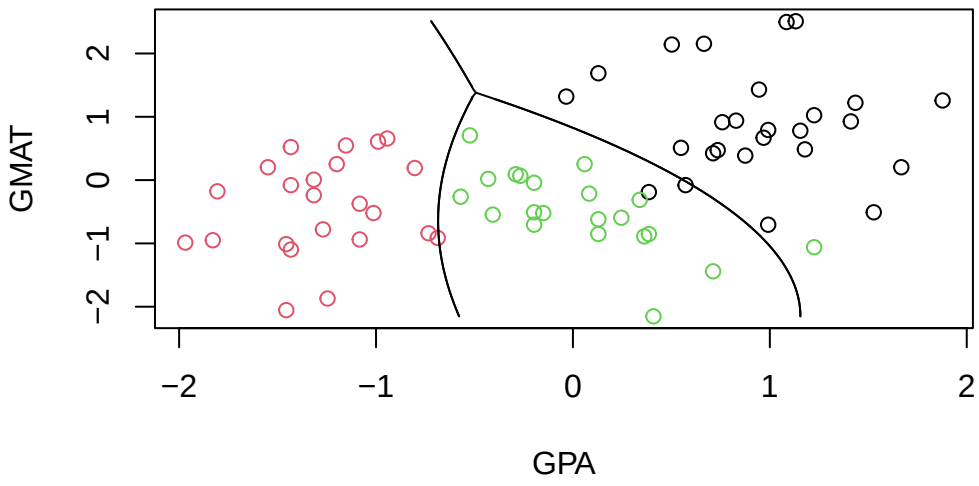
Naive Bayes

```
library(e1071)
```

```
Warning: package 'e1071' was built under R version 4.3.3
```

```
nb.fit <- naiveBayes(Group ~ std_GPA + std_GMAT,
                     data = training_data)

# Decision boundary
# Set up a dense grid and compute posterior prob on the grid
n.grid <- 100
x1.grid <- seq(f = min(training_data[, "std_GPA"]),
               t = max(training_data[, "std_GPA"]),
               l = n.grid)
x2.grid <- seq(f = min(training_data[, "std_GMAT"]),
               t = max(training_data[, "std_GMAT"]),
               l = n.grid)
grid <- expand.grid(x1.grid, x2.grid)
colnames(grid) <- c("std_GPA", "std_GMAT")
pred.grid <- predict(nb.fit, grid, type = "raw")

# Define a function to calculate the decision boundary probabilities
subtract_max_other <- function(row) {
  new_row <- c()
  for (i in 1:3){
    max_other <- max(row[-i])
```

```
    new_row[i] <- row[i] - max_other
  }
  return (new_row)
}

# Apply the function row-wise
probs <- t(apply(pred.grid, 1, subtract_max_other))

# Separate the decision boundary probabilities by Group
prob_1 <- matrix(probs[, 1], nrow = n.grid,
                 ncol = n.grid, byrow = F)
prob_2 <- matrix(probs[, 2], nrow = n.grid,
                 ncol = n.grid, byrow = F)
prob_3 <- matrix(probs[, 3], nrow = n.grid,
                 ncol = n.grid, byrow = F)

# Plot the training data
plot(training_data[, 1:2], col = training_data$Group,
     xlab("GPA"), ylab("GMAT"),
     cex = 1, cex.lab = 1)

# Add the decision boundaries to the plot
contour(x1.grid, x2.grid, prob_1,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
contour(x1.grid, x2.grid, prob_2,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
contour(x1.grid, x2.grid, prob_3,
        levels = 0, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
```




```
# Compute confusion matrix and misclassification rate for training data
train_pred <- predict(nb.fit, training_data)
train_confusion <- table(Predicted = train_pred,
                         Actual = training_data$Group)
train_error_rate <- 1 - sum(diag(train_confusion)) /
  sum(train_confusion)

# Compute confusion matrix and misclassification rate for test data
test_pred <- predict(nb.fit, test_data)
test_confusion <- table(Predicted = test_pred,
                        Actual = test_data$Group)
test_error_rate <- 1 - sum(diag(test_confusion)) /
  sum(test_confusion)

cat("Confusion Matrix (Training Data | Test Data):\n")
```

```
Confusion Matrix (Training Data | Test Data):
```

```
print(cbind(train_confusion,test_confusion))
```

```
   1  2  3 1 2 3
```

```
1 24  0  1 4 0 0
2  0 23  0 0 2 0
3  2  0 20 1 3 5
```

The plot demonstrates clear distinctions between the three different labels. The correctly classifying all the data points except for only two observations within the training dataset.

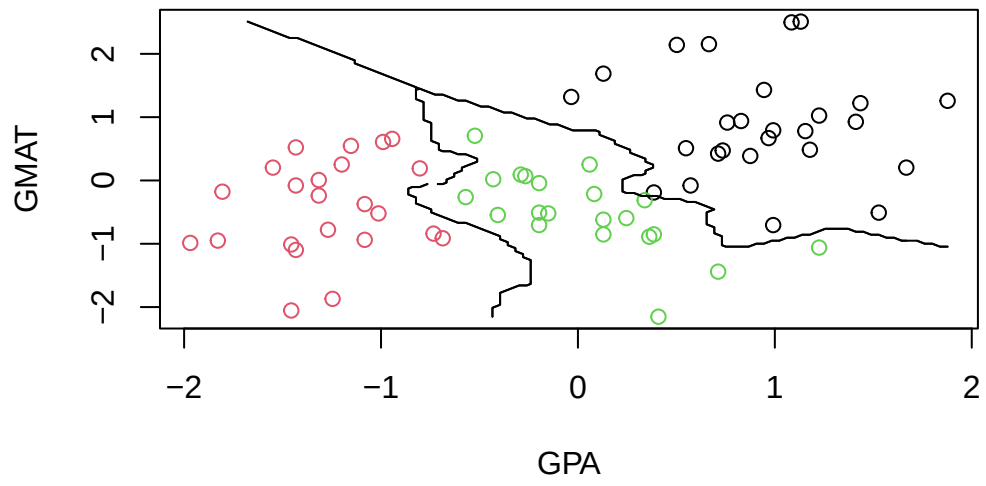
The confusion matrix for the training data shows very good performance. There is only two misclassification in total. The misclassification rate is relatively low (4.29%), suggesting that the model performs reasonably well on the training data. In the test data, there are more misclassifications compared to the training data. The misclassification rate is higher (26.67%) compared to the training data. This suggests that the model might have a problem of overfitting.

KNN

```
library(class)

k_values <- seq(1, 20, by = 2) # Set of possible k values
test_errors <- numeric(length(k_values)) # Vector to store test errors

for (i in 1:length(k_values)) {
  set.seed(3284) # Set seed for reproducibility
  test_errors[i] <- mean(sapply(k_values[i], function(k) {
    test_pred <- knn(train = training_data[, c("std_GPA", "std_GMAT")],
                     test = test_data[, c("std_GPA", "std_GMAT")],
                     cl = training_data$Group, k = k)
    return(mean(test_pred != test_data$Group))
  }))
}

# Find the optimal k value with the lowest cross-validation error
optimal_k <- k_values[which.min(test_errors)]

# Apply KNN with the optimal k value
knn_fit <- knn(train = training_data[, c("std_GPA", "std_GMAT")],
               test = test_data[, c("std_GPA", "std_GMAT")],
               cl = training_data$Group, k = optimal_k)

# Compute confusion matrix and misclassification rate for training data
knn_train_pred <- knn(train = training_data[, c("std_GPA", "std_GMAT")],
                      test = training_data[, c("std_GPA", "std_GMAT")],
```

```
                      cl = training_data$Group, k = optimal_k)
knn_train_confusion <- table(Predicted = knn_train_pred,
                             Actual = training_data$Group)
knn_train_error_rate <- 1 - sum(diag(knn_train_confusion)) /
  sum(knn_train_confusion)


# Compute confusion matrix and misclassification rate for test data
knn_confusion <- table(Predicted = knn_fit,
                       Actual = test_data$Group)
knn_error_rate <- 1 - sum(diag(knn_confusion)) /
  sum(knn_confusion)

# Decision boundary
# Set up a dense grid and compute prob on the grid
n.grid <- 100
x1.grid <- seq(f = min(training_data[, "std_GPA"]),
               t = max(training_data[, "std_GPA"]),
               l = n.grid)
x2.grid <- seq(f = min(training_data[, "std_GMAT"]),
               t = max(training_data[, "std_GMAT"]),
               l = n.grid)
grid <- expand.grid(x1.grid, x2.grid)
colnames(grid) <- c("std_GPA", "std_GMAT")

knn.grid <- knn(training_data[, c("std_GPA", "std_GMAT")],
                grid, training_data$Group,
                k = optimal_k, prob = TRUE)

prob <- attr(knn.grid, "prob")
prob_1 <- ifelse(knn.grid == "1", prob, 1 - prob)
prob_1 <- matrix(prob_1, nrow = n.grid,
                 ncol = n.grid, byrow = F)

prob_2 <- ifelse(knn.grid == "2", prob, 1 - prob)
prob_2 <- matrix(prob_2, nrow = n.grid,
                 ncol = n.grid, byrow = F)

prob_3 <- ifelse(knn.grid == "3", prob, 1 - prob)
prob_3 <- matrix(prob_3, nrow = n.grid,
                 ncol = n.grid, byrow = F)
```

```r
# Plot the training data
plot(training_data[, 1:2],
     col = training_data$Group, xlab("GPA"), ylab("GMAT"),
     cex = 1, cex.lab = 1)

# Add the decision boundaries to the plot
contour(x1.grid, x2.grid, prob_1,
        levels = 0.5, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
contour(x1.grid, x2.grid, prob_2,
        levels = 0.5, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
contour(x1.grid, x2.grid, prob_3,
        levels = 0.5, labels = "",
        xlab = "", ylab = "",
        main = "", add = T)
```




```r
cat("Confusion Matrix (Training Data | Test Data):\n")
```

```
Confusion Matrix (Training Data | Test Data):
```

```
print(cbind(knn_train_confusion, knn_confusion))
```

```
   1  2  3 1 2 3
1 26  0  0 5 0 0
2  0 23  0 0 4 0
3  0  0 21 0 1 5
```

The decision boundaries are depicted by the black curve, the model successfully classifies all data points within the training dataset without any misclassifications.

The confusion matrix for the training data using KNN shows excellent performance. There are no misclassifications, the misclassification rate is 0%, indicating perfect classification on the training data. In the test data, there is one misclassification in Class 2. The misclassification rate is 6.67%, which is still relatively low.

The optimal value of $k = 1$ results in an overly complex model that perfectly fits the training data but does not generalize well to new, unseen data, which indicates overfitting. This is probability due to the fact that there are only 15 observations in the test data and we didn't consider the training error at all when finding the optimal value of k.

The Best Model

The QDA model is recommended, as it has a relatively small test error (13.33%) without the issue of the overfitting.

Logistic Regression

Use the German credit dataset available on from https://www.biz.uiowa.edu/faculty/jledolter/DataMining/data
You can read more about the data at https://archive.ics.uci.edu/dataset/144/statlog+german+credit+data.
We take Default as the response, the other variables as predictors, and all the data as training data.

In R

Logistic regression

```r
# Read data
credit <- read.csv("germancredit.csv")

# Modify the data
credit <- credit %>%
  mutate_if(sapply(credit, is.character), ~as.factor(.)) %>%
  mutate_if(sapply(credit, is.integer), ~as.numeric(.))
credit$Default <- as.factor(credit$Default)

# Standardize numeric columns
numeric_columns <- c("duration", "amount", "installment",
                     "residence", "age", "cards", "liable")
credit[numeric_columns] <- lapply(credit[numeric_columns],
                                  function(x) (x - mean(x)) / sd(x))

# Fit the full model
model_full <- glm(Default ~ ., data = credit,
                  family = binomial)

glm.probs <- predict(model_full, type = "response")

predicted <- rep(0, nrow(credit))
predicted[glm.probs > .5] = 1 # predicted label
predicted <- as.factor(predicted)

# Confusion matrix
confusion <- confusionMatrix(predicted, credit$Default)

# Extract metrics
error_rate <- 1 - confusion$overall["Accuracy"]
sensitivity <- confusion$byClass["Sensitivity"]
specificity <- confusion$byClass["Specificity"]

# Print the metrics
cat("Error Rate:", error_rate, "\n")
```

```
Error Rate: 0.214
```

```r
cat("Sensitivity:", sensitivity, "\n")
```

```
Sensitivity: 0.8942857
```

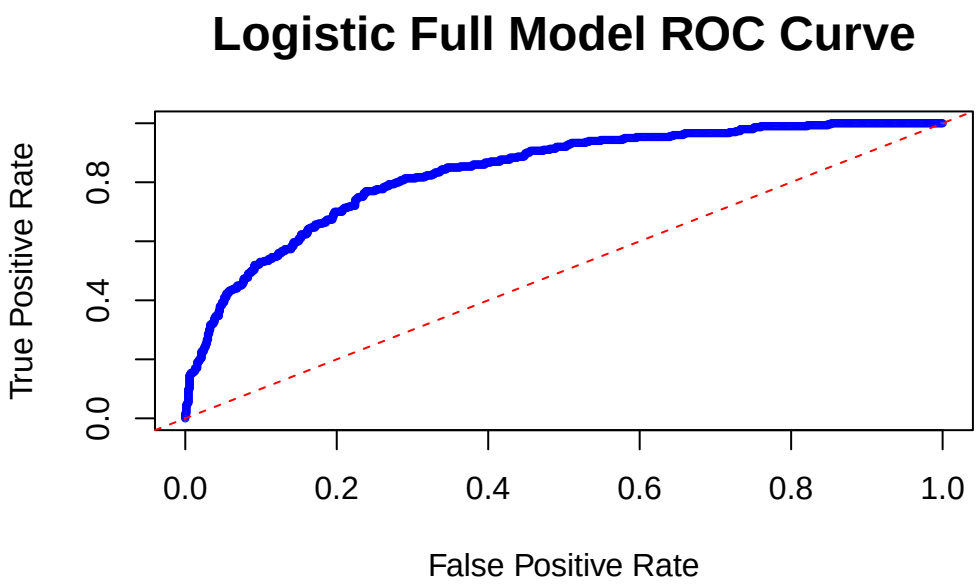
```r
cat("Specificity:", specificity, "\n")
```

```
Specificity: 0.5333333
```

```r
# Create ROC curve
roc_curve <- roc(credit$Default, glm.probs)

# Calculate AUC
auc(roc_curve)
```

```
Area under the curve: 0.8338
```

```r
# ROC curve plot
plot(1-roc_curve$specificities,
     roc_curve$sensitivities,
     main = "Logistic Full Model ROC Curve",
     col = "blue", cex = 0.5, lwd = 0.5,
     xlab = "False Positive Rate",
     ylab = "True Positive Rate",
     xlim = c(0, 1), ylim = c(0, 1),
     cex.main = 1.5,
     cex.lab = 1,
     cex.axis = 1)
# Add diagonal reference line for comparison
abline(a = 0, b = 1, lty = 2, col = "red")
```



Defined LOOCV

```
n <- nrow(credit)
error_count <- 0

for (i in 1:n) {
  # Create a subset for training (excluding one observation)
  train_subset <- credit[-i, ]

  # Fit the model on the subset
  model <- glm(Default ~ .,
               data = train_subset,
               family = binomial)

  # Predict on the left-out observation
  prediction <- predict(model,
                        newdata = credit[i, ],
                        type = "response")

  # Classify the observation
  if (prediction > 0.5) {
    predicted_class <- 1
```

```
  } else {
    predicted_class <- 0
  }

  # Check if the prediction matches the true class
  if (predicted_class != credit$Default[i]) {
    error_count <- error_count + 1
  }
}

# Calculate test error rate
test_error_rate <- error_count / n
cat("Test Error Rate (LOOCV):", test_error_rate, "\n")
```

```
Test Error Rate (LOOCV): 0.249
```

Package LOOCV

```
cost <- function(r, pi = 0) mean(abs(r-pi) > 0.5)
# Perform LOOCV
cv.err <- cv.glm(credit, model_full, cost, K = nrow(credit))
cv.err$delta[1]
```

```
[1] 0.249
```

```
# Print the test error rate
cat("Test Error Rate (LOOCV):", cv.err$delta[1], "\n")
```

```
Test Error Rate (LOOCV): 0.249
```

LDA

```
# Fit lda model
model_lda <- lda(Default ~ ., data = credit)

lda.pred <- predict(model_lda, credit)

lda.class <- lda.pred$class
```

```r
# Confusion matrix
confusion <- confusionMatrix(lda.class, credit$Default)

# Extract metrics
error_rate <- 1 - confusion$overall["Accuracy"]
sensitivity <- confusion$byClass["Sensitivity"]
specificity <- confusion$byClass["Specificity"]

# Print the metrics
cat("Error Rate:", error_rate, "\n")
```

```
Error Rate: 0.223
```

```r
cat("Sensitivity:", sensitivity, "\n")
```

```
Sensitivity: 0.8785714
```

```r
cat("Specificity:", specificity, "\n")
```

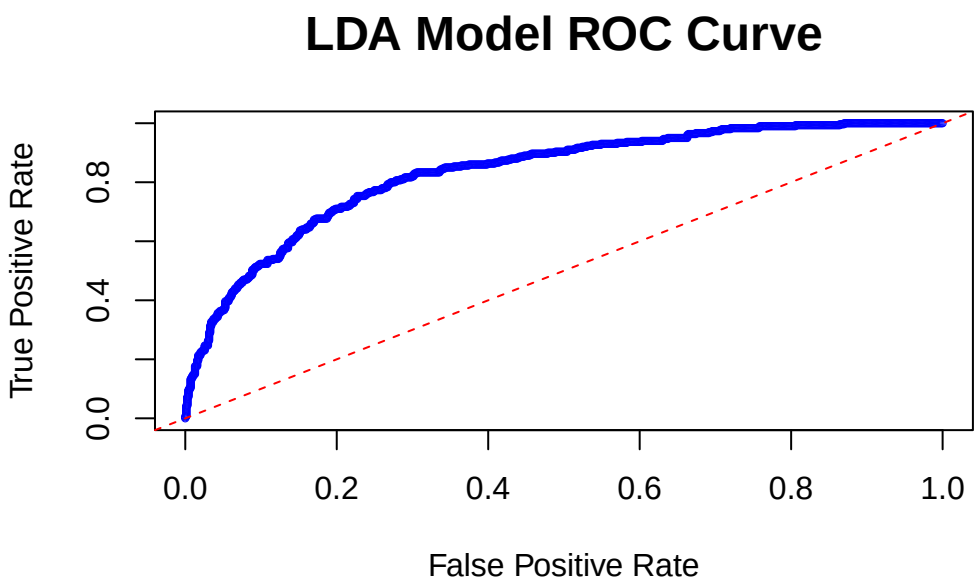
```
Specificity: 0.54
```

```r
# Create ROC curve
lda.probs <- lda.pred$posterior[, 1]
roc_curve <- roc(credit$Default, lda.probs)

# Calculate AUC
auc(roc_curve)
```

```
Area under the curve: 0.8322
```

```r
# ROC curve plot
plot(1-roc_curve$specificities,
     roc_curve$sensitivities,
     main = "LDA Model ROC Curve",
     col = "blue", cex = 0.5, lwd = 0.5,
     xlab = "False Positive Rate",
     ylab = "True Positive Rate",
     xlim = c(0, 1), ylim = c(0, 1),
     cex.main = 1.5,
```

```
     cex.lab = 1,
     cex.axis = 1)
# Add diagonal reference line for comparison
abline(a = 0, b = 1, lty = 2, col = "red")
```




```
# LOOCV
n <- nrow(credit)
error_count <- 0

for (i in 1:n) {
  # Create a subset for training (excluding one observation)
  train_subset <- credit[-i, ]

  # Fit the model on the subset
  model <- lda(Default ~ ., data = train_subset)

  # Predict on the left-out observation
  prediction <- predict(model, newdata = credit[i, ])$class

  # Check if the prediction matches the true class
  if (prediction != credit$Default[i]) {
    error_count <- error_count + 1
```

```
  }
}

# Calculate test error rate
test_error_rate <- error_count / n
cat("Test Error Rate (LOOCV):", test_error_rate, "\n")
```

```
Test Error Rate (LOOCV): 0.242
```

QDA

```
model_qda <- qda(Default ~ ., data = credit)

qda.pred <- predict(model_qda, credit)

qda.class <- qda.pred$class

# Confusion matrix
confusion <- confusionMatrix(qda.class, credit$Default)

# Extract metrics
error_rate <- 1 - confusion$overall["Accuracy"]
sensitivity <- confusion$byClass["Sensitivity"]
specificity <- confusion$byClass["Specificity"]

# Print the metrics
cat("Error Rate:", error_rate, "\n")
```

```
Error Rate: 0.177
```

```
cat("Sensitivity:", sensitivity, "\n")
```

```
Sensitivity: 0.8471429
```

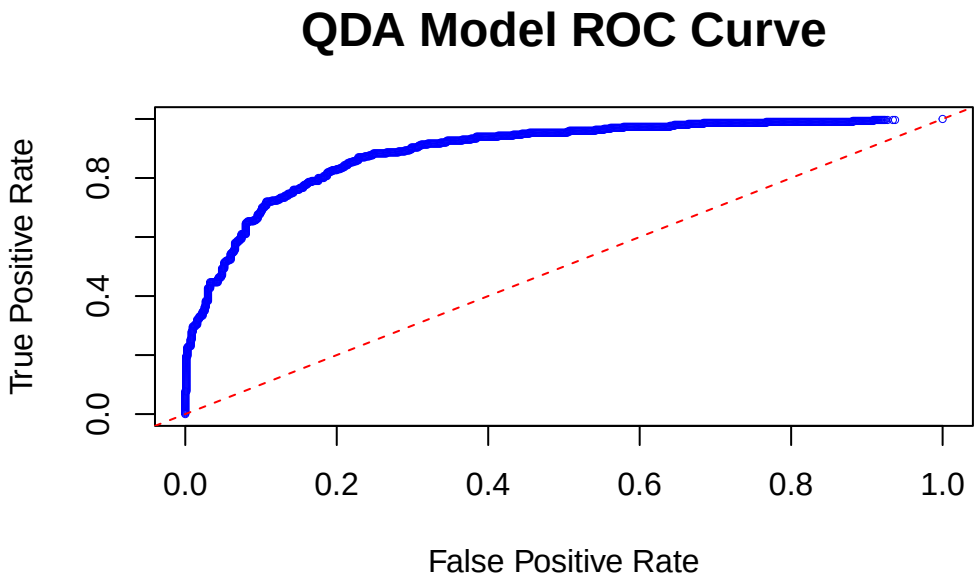
```
cat("Specificity:", specificity, "\n")
```

```
Specificity: 0.7666667
```

```r
# Create ROC curve
qda.probs <- qda.pred$posterior[, 1]
roc_curve <- roc(credit$Default, qda.probs)

# Calculate AUC
auc(roc_curve)
```

```
Area under the curve: 0.8907
```

```r
# ROC curve plot
plot(1-roc_curve$specificities,
     roc_curve$sensitivities,
     main = "QDA Model ROC Curve",
     col = "blue", cex = 0.5, lwd = 0.5,
     xlab = "False Positive Rate",
     ylab = "True Positive Rate",
     xlim = c(0, 1), ylim = c(0, 1),
     cex.main = 1.5,
     cex.lab = 1,
     cex.axis = 1)
# Add diagonal reference line for comparison
abline(a = 0, b = 1, lty = 2, col = "red")
```



```r
n <- nrow(credit)
error_count <- 0

for (i in 1:n) {
  if (i != 204){ # observation 204 has a multicollinearity problem
      # Create a subset for training (excluding one observation)
    train_subset <- credit[-i, ]

    # Fit the model on the subset
    model <- qda(Default ~ ., data = train_subset)

    # Predict on the left-out observation
    prediction <- predict(model, newdata = credit[i, ])$class

    # Check if the prediction matches the true class
    if (prediction != credit$Default[i]) {
      error_count <- error_count + 1
    }
  }
}

# Calculate test error rate
test_error_rate <- error_count / (n-1)
cat("Test Error Rate (LOOCV):", test_error_rate, "\n")
```

```
Test Error Rate (LOOCV): 0.2832833
```

KNN

```r
# Read data
credit <- read.csv("germancredit.csv")

# Modify the data
credit <- credit %>%
  mutate_if(sapply(credit, is.character), ~as.factor(.)) %>%
  mutate_if(sapply(credit, is.integer), ~as.numeric(.))
credit$Default <- as.factor(credit$Default)

# First change the character variables to numeric
credit[, -1] <- credit[, -1] %>%
  mutate_if(sapply(credit[, -1], is.factor), ~as.numeric(.))
```

```r
predictors <- colnames(credit)[-1]

k_values <- seq(1, 100, by = 1) # Set of possible k values
test_errors <- numeric(length(k_values)) # Vector to store test errors

# Get test errors in LOOCV
for (i in 1:length(k_values)) {
  test_errors[i] <- mean(sapply(k_values[i], function(k) {
    error_count <- 0
    for (j in 1:nrow(credit)){
      train_subset <- credit[-j, ]
      test_subset <- credit[j, ]
      set.seed(3284) # Set seed for reproducibility
      test_pred <- knn(train = train_subset[, predictors],
                       test = test_subset[, predictors],
                       cl = as.numeric(as.character(train_subset$Default)),
                       k = k)
      # Check if the prediction matches the true class
      if (test_pred != test_subset$Default) {
        error_count <- error_count + 1
      }
    }
    return(error_count / nrow(credit))
  }))
}

# Find the optimal k value with the lowest LOOCV test error
optimal_k <- k_values[which.min(test_errors)]

set.seed(3284)
# Apply KNN with the optimal k value on the entire data set
knn_pred <- knn(train = credit[, predictors],
                test = credit[, predictors],
                cl = credit$Default,
                k = optimal_k,
                prob = TRUE)

# Confusion matrix
confusion <- confusionMatrix(knn_pred, credit$Default)

# Extract metrics
error_rate <- 1 - confusion$overall["Accuracy"]
```

```r
sensitivity <- confusion$byClass["Sensitivity"]
specificity <- confusion$byClass["Specificity"]

# Print the metrics
cat("Error Rate:", error_rate, "\n")
```

```
Error Rate: 0.288
```

```r
cat("Sensitivity:", sensitivity, "\n")
```

```
Sensitivity: 0.9885714
```

```r
cat("Specificity:", specificity, "\n")
```

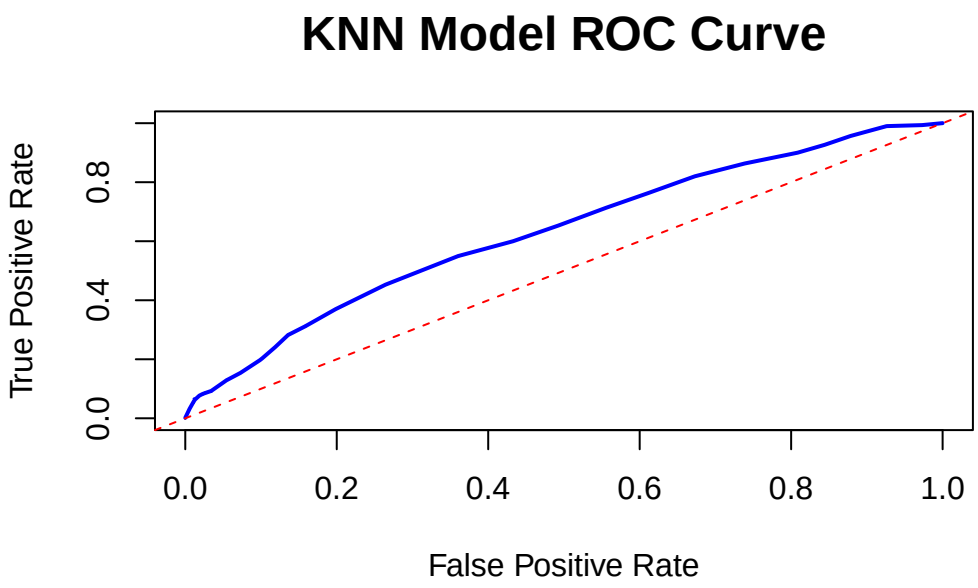
```
Specificity: 0.06666667
```

```r
# Create ROC curve
probs <- attr(knn_pred, "prob")
knn.probs <- ifelse(knn_pred == "1", probs, 1 - probs)

roc_curve <- roc(credit$Default, knn.probs)

# Calculate AUC
auc(roc_curve)
```

```
Area under the curve: 0.6296
```

```r
plot(1-roc_curve$specificities,
     roc_curve$sensitivities,
     main = "KNN Model ROC Curve",
     col = "blue", cex = 0.1, lwd = 0.5,
     xlab = "False Positive Rate",
     ylab = "True Positive Rate",
     xlim = c(0, 1), ylim = c(0, 1),
     cex.main = 1.5,
     cex.lab = 1,
     cex.axis = 1)
smooth_line <- smooth.spline(1-roc_curve$specificities,
                             roc_curve$sensitivities)
```

```
lines(smooth_line, col = "blue", lwd = 2)
# Add diagonal reference line for comparison
abline(a = 0, b = 1, lty = 2, col = "red")
```



Proposed model

```
# Read data
credit <- read.csv("germancredit.csv")

# Modify the data
credit <- credit %>%
  mutate_if(sapply(credit, is.character), ~as.factor(.)) %>%
  mutate_if(sapply(credit, is.integer), ~as.numeric(.))
credit$Default <- as.factor(credit$Default)

# Standardize numeric columns
numeric_columns <- c("duration", "amount", "installment",
                     "residence", "age", "cards", "liable")
credit[numeric_columns] <- lapply(credit[numeric_columns],
                                  function(x) (x - mean(x)) / sd(x))
```

```r
# Fit the proposed model
model_sub <- glm(Default ~ checkingstatus1 + duration + history +
                   purpose + amount + savings + installment +
                   status + others + otherplans + foreign,
                 data = credit, family = binomial)

glm.probs <- predict(model_sub, type = "response")

predicted <- rep(0, nrow(credit))
predicted[glm.probs > .5] = 1 # predicted label
predicted <- as.factor(predicted)

# Confusion matrix
confusion <- confusionMatrix(predicted, credit$Default)

# Extract metrics
error_rate <- 1 - confusion$overall["Accuracy"]
sensitivity <- confusion$byClass["Sensitivity"]
specificity <- confusion$byClass["Specificity"]

# Print the metrics
cat("Error Rate:", error_rate, "\n")
```

```
Error Rate: 0.216
```

```r
cat("Sensitivity:", sensitivity, "\n")
```

```
Sensitivity: 0.8914286
```

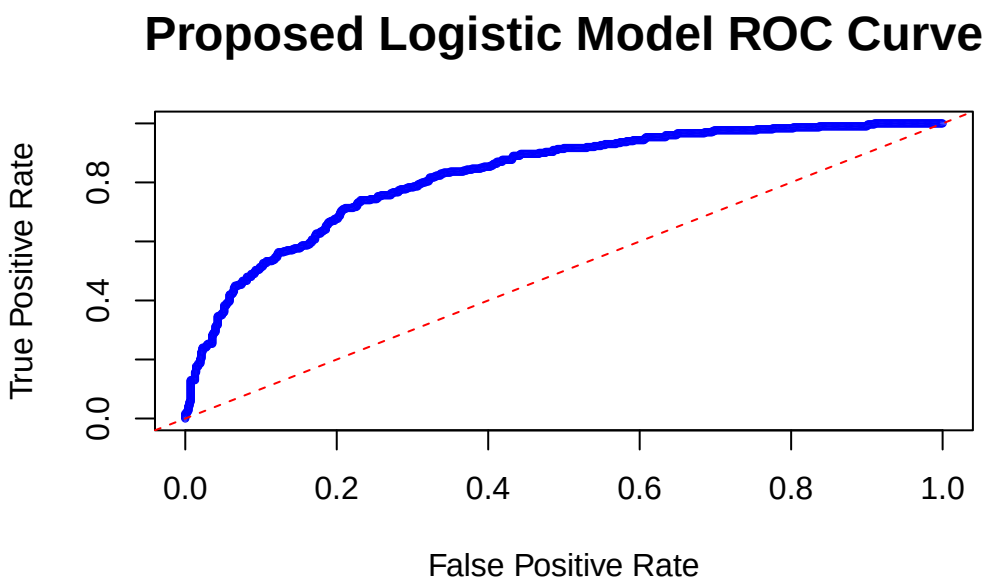
```r
cat("Specificity:", specificity, "\n")
```

```
Specificity: 0.5333333
```

```r
# Create ROC curve
roc_curve <- roc(credit$Default, glm.probs)

# Calculate AUC
auc(roc_curve)
```

```
Area under the curve: 0.8239
```

```r
# ROC curve plot
plot(1-roc_curve$specificities,
     roc_curve$sensitivities,
     main = "Proposed Logistic Model ROC Curve",
     col = "blue", cex = 0.5, lwd = 0.5,
     xlab = "False Positive Rate",
     ylab = "True Positive Rate",
     xlim = c(0, 1), ylim = c(0, 1),
     cex.main = 1.5,
     cex.lab = 1,
     cex.axis = 1)
# Add diagonal reference line for comparison
abline(a = 0, b = 1, lty = 2, col = "red")
```




```r
# LOOCV test error
cost <- function(r, pi = 0) mean(abs(r-pi) > 0.5)
# Perform LOOCV
cv.err <- cv.glm(credit, model_sub, cost, K = nrow(credit))
cv.err$delta[1]
```

```
[1] 0.235
```

```r
# Print the test error rate
cat("Test Error Rate (LOOCV):", cv.err$delta[1], "\n")
```

```
Test Error Rate (LOOCV): 0.235
```

Ridge and Lasso Regression

Consider the prostate cancer dataset. Take `psa` in the prostate cancer dataset as the quantitative response variable. Among the predictors, `vesinv` is treated as a qualitative variable and `gleason` as a quantitative variable. Take all the data as training data. For all the models below, leave-one-out cross- validation (LOOCV) is used to compute the estimated test MSE.

In R

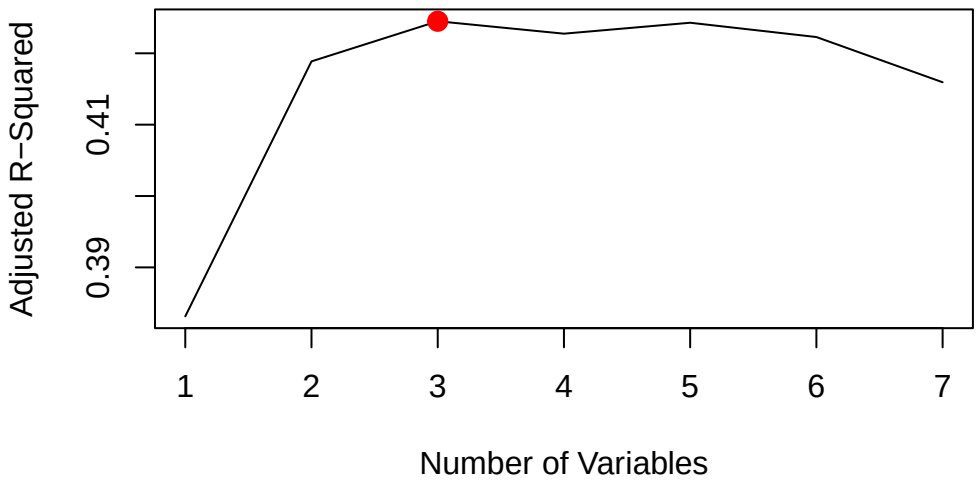
Linear regression

```r
library(boot)

# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)

prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Define the linear regression model
full_model <- glm(psa ~ ., data = prostate_cancer)

# Perform LOOCV
loocv_result <- cv.glm(prostate_cancer, full_model)$delta[1]

# Print the result
loocv_result
```

```
[1] 1218.357
```

Best-subset selection

```r
library(leaps)
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)

prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

totpred <- ncol(prostate_cancer) - 1
fit.full <- regsubsets(psa ~ ., prostate_cancer, nvmax = totpred)

# Adjusted R^2
fit.summary <- summary(fit.full)
#which.max(fit.summary$adjr2) #3

# Plot model fit adjusted r-squared for best model of each size against size
plot(fit.summary$adjr2,
     xlab = "Number of Variables",
     ylab = "Adjusted R-Squared",
     type = "l")

points(3, fit.summary$adjr2[3],
       col = "red", cex = 2, pch = 20)
```



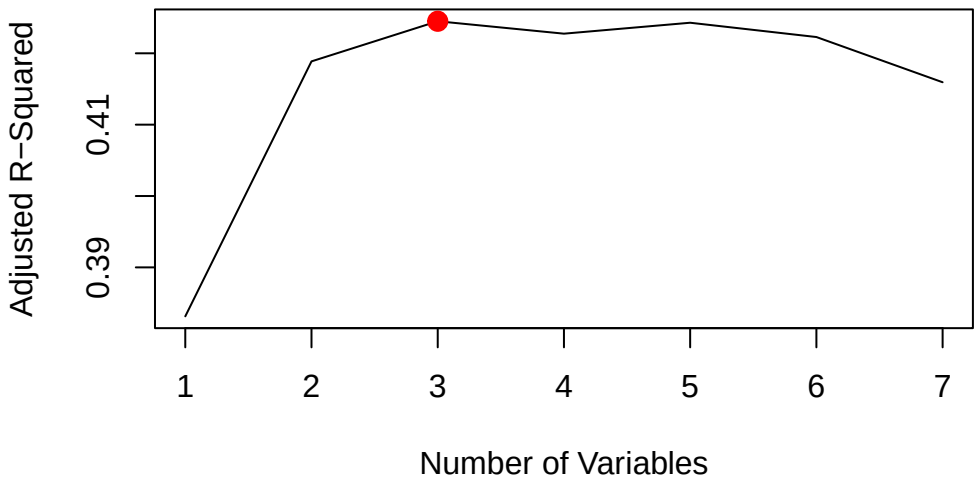

```
# Get coefficients of best model for a given size
best_subset_coef <- coef(fit.full, 3)

# Define the best linear regression model
best_subset_model <- glm(psa ~ cancervol + vesinv + gleason,
                         data = prostate_cancer)

# Perform LOOCV
loocv_result <- cv.glm(prostate_cancer, best_subset_model)$delta[1]

# Print the result
loocv_result
```

```
[1] 1084.374
```

Forward stepwise selection

```
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)
```

```r
prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Forward stepwise selection
totpred <- ncol(prostate_cancer) - 1
fit.fwd = regsubsets(psa ~ ., data = prostate_cancer,
                     nvmax = totpred,
                     method = "forward")

# Adjusted R^2
fit.summary <- summary(fit.fwd)
#which.max(fit.summary$adjr2) #3

# Plot model fit adjusted r-squared for best model of each size against size
plot(fit.summary$adjr2,
     xlab = "Number of Variables",
     ylab = "Adjusted R-Squared",
     type = "l")

points(3, fit.summary$adjr2[3],
       col = "red", cex = 2, pch = 20)
```



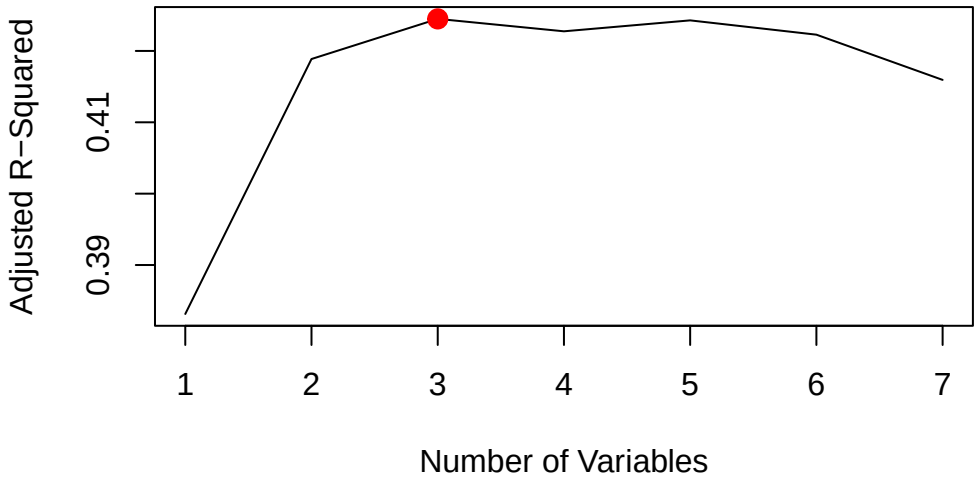
```r
# Get coefficients of best model for a given size
fwd_coeff <- coef(fit.fwd, 3)

# Define the best linear regression model
fwd_model <- glm(psa ~ cancervol + vesinv + gleason,
                 data = prostate_cancer)

# Perform LOOCV
loocv_result <- cv.glm(prostate_cancer, fwd_model)$delta[1]

# Print the result
loocv_result
```

```
[1] 1084.374
```

Backward stepwise selection

```r
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)

prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Forward stepwise selection
totpred <- ncol(prostate_cancer) - 1
fit.bwd = regsubsets(psa ~ ., data = prostate_cancer,
                     nvmax = totpred,
                     method = "backward")

# Adjusted R^2
fit.summary <- summary(fit.bwd)
#which.max(fit.summary$adjr2) #3

# Plot model fit adjusted r-squared for best model of each size against size
plot(fit.summary$adjr2,
     xlab = "Number of Variables",
     ylab = "Adjusted R-Squared",
     type = "l")
```

```
points(3, fit.summary$adjr2[3],
       col = "red", cex = 2, pch = 20)
```




```
# Get coefficients of best model for a given size
bwd_coeff <- coef(fit.bwd, 3)

# Define the best linear regression model
bwd_model <- glm(psa ~ cancervol + vesinv + gleason,
                 data = prostate_cancer)

# Perform LOOCV
loocv_result <- cv.glm(prostate_cancer, bwd_model)$delta[1]

# Print the result
loocv_result
```

```
[1] 1084.374
```

Ridge regression

```r
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)

prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Ridge regression
y <- prostate_cancer$psa
x <- model.matrix(psa ~ ., prostate_cancer) # with intercept

# LOOCV
cv.out <- cv.glmnet(x, y, nfolds = length(y), alpha = 0)

# Find the best value of lambda
ridge_bestlam <- cv.out$lambda.min
ridge_bestlam
```

```
[1] 19.60679
```

```r
# Test MSE for the best value of lambda
test_error <- c()
for (i in 1:nrow(x)){
  train_x <- x[-i, ]
  train_y <- y[-i]
  test_x <- x[i, ]
  test_y <- y[i]
  ridge.mod <- glmnet(train_x, train_y, alpha = 0)
  test_pred <- predict(ridge.mod, s = ridge_bestlam,
                       newx = test_x)
  test_error[i] <- (test_pred - test_y)^2
}
test_MSE <- mean(test_error)
test_MSE
```

```
[1] 1137.31
```

```r
# Refit the model on the full dataset
out <- glmnet(x, y, alpha = 0)
ridge_coeff <- predict(out, type = "coefficients",
                       s = ridge_bestlam)
```

Lasso regression

```
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)

prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# lasso regression
y <- prostate_cancer$psa
x <- model.matrix(psa ~ ., prostate_cancer) # with intercept

# LOOCV
cv.out <- cv.glmnet(x, y, nfolds = length(y), alpha = 1)

# Find the best value of lambda
lasso_bestlam <- cv.out$lambda.min
lasso_bestlam
```

```
[1] 2.715317
```

```
# Test MSE for the best value of lambda
test_error <- c()
for (i in 1:nrow(x)){
  train_x <- x[-i, ]
  train_y <- y[-i]
  test_x <- x[i, ]
  test_y <- y[i]
  lasso.mod <- glmnet(train_x, train_y, alpha = 1)
  test_pred <- predict(lasso.mod, s = lasso_bestlam,
                       newx = test_x)
  test_error[i] <- (test_pred - test_y)^2
}
test_MSE <- mean(test_error)
test_MSE
```

```
[1] 1153.383
```

```r
# Refit the model on the full dataset
out <- glmnet(x, y, alpha = 1)
lasso_coeff <- predict(out, type = "coefficients", s = lasso_bestlam)
```

Summary

```r
# The coefficients table
# The linear regression full model
coefs_full <- summary(full_model)$coefficients[, 1]
coefs_full <- data.frame(
  Estimate = round(coefs_full, 2)
)

# The best-subset selection
best_subset_coef <- as.data.frame(best_subset_coef)

# The forward stepwise selection
fwd_coeff <- as.data.frame(fwd_coeff)

# The backward stepwise selection
bwd_coeff <- as.data.frame(bwd_coeff)

# The ridge regression
ridge_coeff <- as.data.frame(as.matrix(ridge_coeff))
ridge_coeff <- ridge_coeff[-2, ,drop = FALSE]
ridge_coeff$s1 <- round(ridge_coeff$s1, 2)
rownames(ridge_coeff)[1] <- "(Intercept)"

# The lasso logistic model
lasso_coeff <- as.data.frame(as.matrix(lasso_coeff))
lasso_coeff <- lasso_coeff[-2, ,drop = FALSE]
lasso_coeff$s1 <- round(lasso_coeff$s1, 2)
rownames(lasso_coeff)[1] <- "(Intercept)"

df <- data.frame(matrix(nrow = 8, ncol = 6))
df$FullModel <- coefs_full$Estimate
df$BestSubset <- c(-44.18, 2.25, NA, NA, NA, 21.88, NA, 6.90)
df$Forward <- c(-44.18, 2.25, NA, NA, NA, 21.88, NA, 6.90)
df$Backward <- c(-44.18, 2.25, NA, NA, NA, 21.88, NA, 6.90)
df$Ridge <- ridge_coeff$s1
df$Lasso <- lasso_coeff$s1
```

```r
rownames(df) <- rownames(coefs_full)

# Add test error rate to the table
df["TestMSE", ] <- c(1218.357, 1084.374, 1084.374,
                             1084.374, 1137.31, 1153.383)

# Print the coefficients table
df <- df %>% dplyr::select(-X1, -X2, -X3, -X4, -X5, -X6)
kable(df, caption = "Regression Coefficients and Test Error Rate")
```

Table 4: Regression Coefficients and Test Error Rate

	FullModel	BestSubset	Forward	Backward	Ridge	Lasso
(Intercept)	-15.240	-44.180	-44.180	-44.180	-23.93	-24.870
cancervol	2.030	2.250	2.250	2.250	1.32	1.910
weight	0.010	NA	NA	NA	0.01	0.000
age	-0.540	NA	NA	NA	-0.26	0.000
benpros	1.300	NA	NA	NA	0.53	0.000
vesinv1	19.610	21.880	21.880	21.880	15.90	15.170
capspen	1.100	NA	NA	NA	1.57	0.930
gleason	7.060	6.900	6.900	6.900	6.71	4.340
TestMSE	1218.357	1084.374	1084.374	1084.374	1137.31	1153.383

In Python

Linear regression

```python
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import LeaveOneOut
import numpy as np

# Read data
prostate_cancer = pd.read_csv("prostate_cancer.csv")

# Define the predictors and response variable
X = prostate_cancer[['cancervol', 'weight', 'age',
                     'benpros', 'vesinv', 'capspen',
                     'gleason']]
X = sm.add_constant(X)  # Add a constant term
```

```
y = prostate_cancer['psa']

# Initialize LOOCV
loo = LeaveOneOut()
mse_list = []

# Perform LOOCV
for train_index, test_index in loo.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    # Fit the linear regression model
    model = sm.OLS(y_train, X_train).fit()

    y_pred = model.predict(X_test)
    mse = ((y_test - y_pred) ** 2).mean()
    mse_list.append(mse)

# Calculate mean MSE
mean_mse = np.mean(mse_list)

# Print the result
print("Estimated Test MSE (LOOCV):", mean_mse)
```

```
Estimated Test MSE (LOOCV): 1218.357459355035
```

Best-subset selection

```
import pandas as pd
import numpy as np
import itertools
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Read data
prostate_cancer = pd.read_csv("prostate_cancer.csv")

dummies = pd.get_dummies(prostate_cancer[['vesinv']])

y = prostate_cancer.psa
```

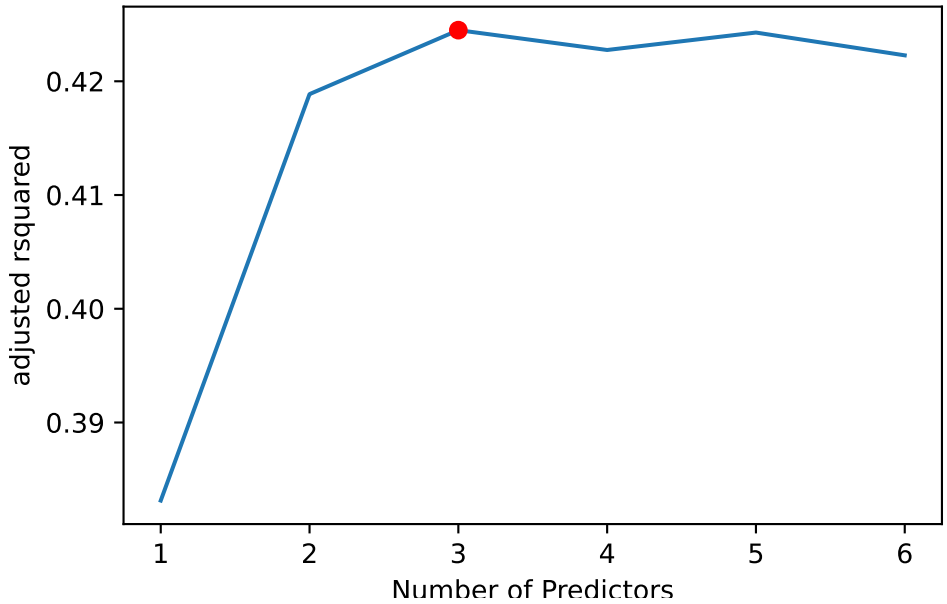
```
# Drop the column with the independent variable (Salary), and columns for which we created du
X_ = prostate_cancer.drop(['psa', 'vesinv', 'subject'], axis=1).astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['vesinv']]], axis=1)

def processSubset(feature_set):
    X_subset = X[list(feature_set)]
    X_subset = sm.add_constant(X_subset)  # Add an intercept term
    # Fit model on feature_set and calculate RSS
    model = sm.OLS(y, X_subset)
    regr = model.fit()
    RSS = ((regr.predict(X_subset) - y) ** 2).sum()
    return {"model":regr, "RSS":RSS}

def getBest(k):

    results = []

    for combo in itertools.combinations(X.columns, k):
        results.append(processSubset(combo))

    # Wrap everything up in a dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    # Return the best model, along with some other useful information about the model
    return best_model

# The best models
models_best = pd.DataFrame(columns=["RSS", "model"])
for i in range(1,7):
    models_best.loc[i] = getBest(i)

models_best.loc[3, "model"].rsquared_adj
```

```
0.4244943463374071
```

```
print(models_best.loc[3, "model"].summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    psa   R-squared:                       0.442
Model:                            OLS   Adj. R-squared:                  0.424
Method:                 Least Squares   F-statistic:                     24.60
Date:                Mon, 20 Jan 2025   Prob (F-statistic):           8.31e-12
Time:                        16:08:08   Log-Likelihood:                -468.50
No. Observations:                  97   AIC:                             945.0
Df Residuals:                      93   BIC:                             955.3
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -44.1849     32.936     -1.342      0.183    -109.589      21.219
cancervol      2.2496      0.519      4.333      0.000       1.219       3.281
gleason        6.8982      4.980      1.385      0.169      -2.992      16.788
vesinv        21.8808      9.588      2.282      0.025       2.841      40.920
==============================================================================
Omnibus:                       97.655   Durbin-Watson:                   0.754
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1219.400
Skew:                           3.242   Prob(JB):                    1.62e-265
Kurtosis:                      19.114   Cond. No.                         126.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
rsquared_adj = models_best.apply(lambda row: row[1].rsquared_adj, axis=1)

plt.plot(rsquared_adj)
plt.plot(rsquared_adj.argmax()+1, rsquared_adj.max(), "or")
plt.xlabel('Number of Predictors')
plt.ylabel('adjusted rsquared')
plt.show()
```




```
plt.close()


X = X[["cancervol", "gleason", "vesinv"]]

model = LinearRegression()

# Perform LOOCV
scores = cross_val_score(model, X, y, cv=len(X), scoring='neg_mean_squared_error')

# Convert the scores to positive values
mse_scores = -scores

# Calculate the mean of the MSE scores
mean_mse = np.mean(mse_scores)

print("Test MSE:", mean_mse)
```

```
Test MSE: 1084.3737605348736
```

Forward stepwise selection

```
import pandas as pd
import numpy as np
import itertools
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Read data
prostate_cancer = pd.read_csv("prostate_cancer.csv")

dummies = pd.get_dummies(prostate_cancer[['vesinv']])

y = prostate_cancer.psa

# Drop the column with the independent variable (Salary), and columns for which we created du
X_ = prostate_cancer.drop(['psa', 'vesinv', 'subject'], axis=1).astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['vesinv']]], axis=1)

def processSubset(feature_set):
    X_subset = X[list(feature_set)]
    X_subset = sm.add_constant(X_subset)  # Add an intercept term
    # Fit model on feature_set and calculate RSS
    model = sm.OLS(y, X_subset)
    regr = model.fit()
    RSS = ((regr.predict(X_subset) - y) ** 2).sum()
    return {"model":regr, "RSS":RSS}

def forward(predictors):

    # Pull out predictors we still need to process
    remaining_predictors = [p for p in X.columns if p not in predictors]

    results = []

    for p in remaining_predictors:
        results.append(processSubset(predictors[1:]+[p]))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
```

```
    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    # Return the best model, along with some other useful information about the model
    return best_model

models_fwd = pd.DataFrame(columns=["RSS", "model"])
predictors = []
for i in range(1,7):
    models_fwd.loc[i] = forward(predictors)
    predictors = models_fwd.loc[i]["model"].model.exog_names

models_fwd.loc[3, "model"].rsquared_adj
```

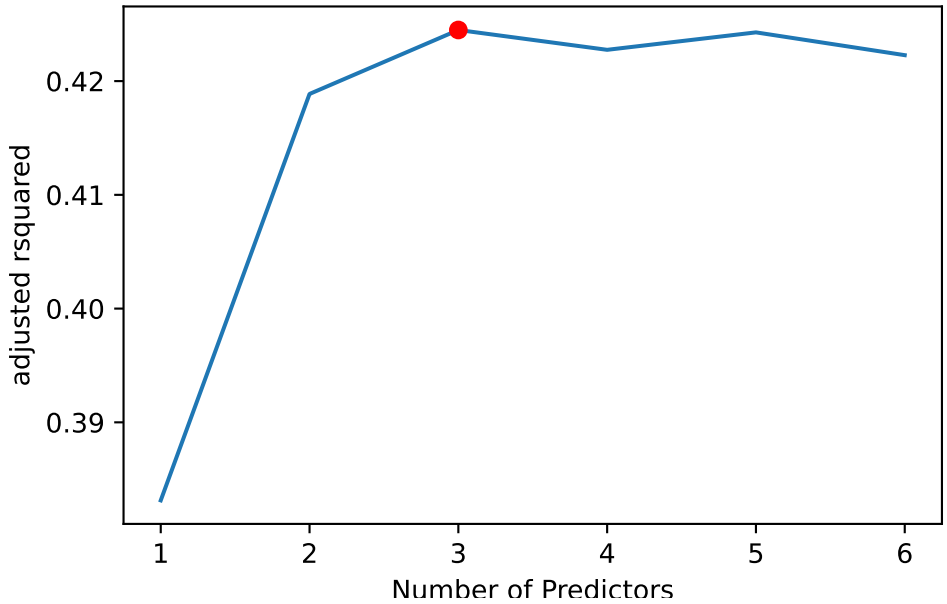
```
0.4244943463374071
```

```
print(models_fwd.loc[3, "model"].summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    psa   R-squared:                       0.442
Model:                            OLS   Adj. R-squared:                  0.424
Method:                 Least Squares   F-statistic:                     24.60
Date:                Mon, 20 Jan 2025   Prob (F-statistic):           8.31e-12
Time:                        16:08:09   Log-Likelihood:                -468.50
No. Observations:                  97   AIC:                             945.0
Df Residuals:                      93   BIC:                             955.3
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -44.1849     32.936     -1.342      0.183    -109.589      21.219
cancervol      2.2496      0.519      4.333      0.000       1.219       3.281
vesinv        21.8808      9.588      2.282      0.025       2.841      40.920
gleason        6.8982      4.980      1.385      0.169      -2.992      16.788
==============================================================================
Omnibus:                       97.655   Durbin-Watson:                   0.754
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1219.400
Skew:                           3.242   Prob(JB):                    1.62e-265
```

```
Kurtosis:                       19.114   Cond. No.                         126.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
rsquared_adj = models_fwd.apply(lambda row: row[1].rsquared_adj, axis=1)

plt.plot(rsquared_adj)
plt.plot(rsquared_adj.argmax()+1, rsquared_adj.max(), "or")
plt.xlabel('Number of Predictors')
plt.ylabel('adjusted rsquared')
plt.show()
```




```
plt.close()

X = X[["cancervol", "gleason", "vesinv"]]

model = LinearRegression()

# Perform LOOCV
scores = cross_val_score(model, X, y, cv=len(X), scoring='neg_mean_squared_error')
```

```
# Convert the scores to positive values
mse_scores = -scores

# Calculate the mean of the MSE scores
mean_mse = np.mean(mse_scores)

print("Test MSE:", mean_mse)
```

```
Test MSE: 1084.3737605348736
```

Backward stepwise selection

```
import pandas as pd
import numpy as np
import itertools
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Read data
prostate_cancer = pd.read_csv("prostate_cancer.csv")

dummies = pd.get_dummies(prostate_cancer[['vesinv']])

y = prostate_cancer.psa

# Drop the column with the independent variable (Salary), and columns for which we created du
X_ = prostate_cancer.drop(['psa', 'vesinv', 'subject'], axis=1).astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['vesinv']]], axis=1)

def processSubset(feature_set):
    X_subset = X[list(feature_set)]
    X_subset = sm.add_constant(X_subset)  # Add an intercept term
    # Fit model on feature_set and calculate RSS
    model = sm.OLS(y, X_subset)
    regr = model.fit()
    RSS = ((regr.predict(X_subset) - y) ** 2).sum()
```

```
    return {"model":regr, "RSS":RSS}

def backward(predictors):

    results = []

    for combo in itertools.combinations(predictors[1:], len(predictors[1:])-1):
        results.append(processSubset(combo))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    # Return the best model, along with some other useful information about the model
    return best_model

models_bwd = pd.DataFrame(columns=["RSS", "model"], index = range(1,len(X.columns)))
X.insert(0, 'const', 1)
predictors = X.columns
while(len(predictors) > 1):
    models_bwd.loc[len(predictors)-1] = backward(predictors)
    predictors = models_bwd.loc[len(predictors)-1]["model"].model.exog_names

models_bwd.loc[4, "model"].rsquared_adj
```

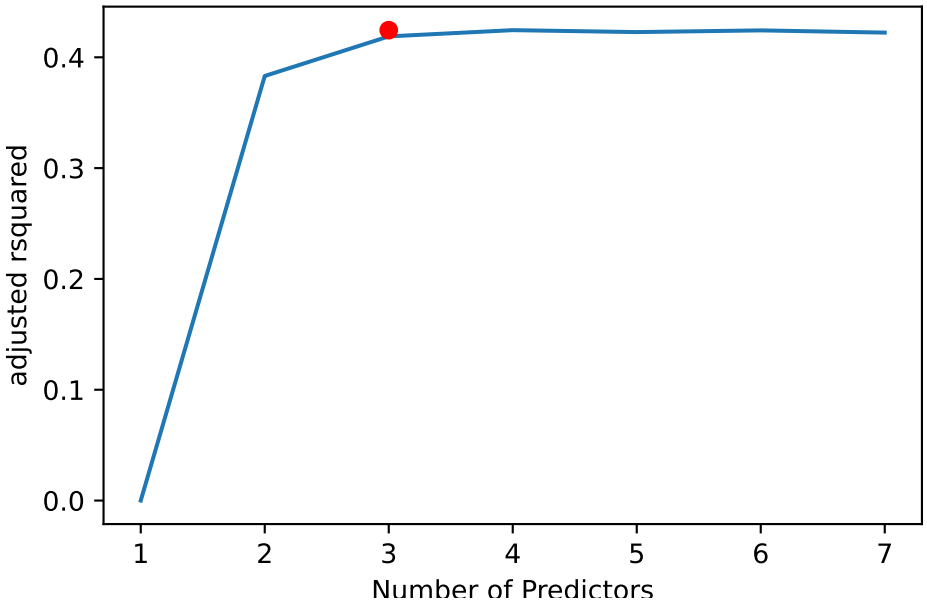
```
0.4244943463374071
```

```
print(models_bwd.loc[4, "model"].summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    psa   R-squared:                       0.442
Model:                            OLS   Adj. R-squared:                  0.424
Method:                 Least Squares   F-statistic:                     24.60
Date:                Mon, 20 Jan 2025   Prob (F-statistic):           8.31e-12
Time:                        16:08:09   Log-Likelihood:                -468.50
No. Observations:                  97   AIC:                             945.0
Df Residuals:                      93   BIC:                             955.3
Df Model:                           3
```

```
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -44.1849     32.936     -1.342      0.183    -109.589      21.219
cancervol      2.2496      0.519      4.333      0.000       1.219       3.281
gleason        6.8982      4.980      1.385      0.169      -2.992      16.788
vesinv        21.8808      9.588      2.282      0.025       2.841      40.920
==============================================================================
Omnibus:                       97.655   Durbin-Watson:                   0.754
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1219.400
Skew:                           3.242   Prob(JB):                    1.62e-265
Kurtosis:                      19.114   Cond. No.                         126.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
rsquared_adj = models_bwd.apply(lambda row: row[1].rsquared_adj, axis=1)

plt.plot(rsquared_adj)
plt.plot(rsquared_adj.argmax(), rsquared_adj.max(), "or")
plt.xlabel('Number of Predictors')
plt.ylabel('adjusted rsquared')
plt.show()
```




```
plt.close()

X = X[["cancervol", "gleason", "vesinv"]]

model = LinearRegression()

# Perform LOOCV
scores = cross_val_score(model, X, y, cv=len(X), scoring='neg_mean_squared_error')

# Convert the scores to positive values
mse_scores = -scores

# Calculate the mean of the MSE scores
mean_mse = np.mean(mse_scores)

print("Test MSE:", mean_mse)
```

```
Test MSE: 1084.3737605348736
```

Ridge regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.model_selection as skm
import sklearn.linear_model as skl
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.metrics import mean_squared_error

# Read data
prostate_cancer = pd.read_csv("prostate_cancer.csv")

dummies = pd.get_dummies(prostate_cancer[['vesinv']])

y = prostate_cancer.psa

# Drop the column with the independent variable (Salary), and columns for which we created du
X_ = prostate_cancer.drop(['psa', 'vesinv', 'subject'], axis=1).astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['vesinv']]], axis=1)

ridgecv = RidgeCV(scoring = 'neg_mean_squared_error')
ridgecv.fit(X, y)
```

```
RidgeCV(scoring='neg_mean_squared_error')
```

```
best_alpha = ridgecv.alpha_

n = 97
test_error = np.zeros(n)

for i in range(n):
    train_x = X.drop(i, axis=0)
    train_y = y.drop(i)
    test_x = X.iloc[i, :].values.reshape(1, -1)
    test_y = y.iloc[i]

    ridge_mod = Ridge(alpha = best_alpha)
    ridge_mod.fit(train_x, train_y)
```

```python
    test_pred = ridge_mod.predict(test_x)
    test_error[i] = (test_pred - test_y)**2
```

```
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
```

```
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
```

```
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
Ridge(alpha=10.0)
```

```
test_MSE = np.mean(test_error)
```

```
print(f"The optimal alpha for Ridge regression is: {best_alpha}")
```

```
The optimal alpha for Ridge regression is: 10.0
```

```
print(f"The test mean squared error is: {test_MSE}")
```

```
The test mean squared error is: 1187.5512136180225
```

```
# The coefficients
ridge = Ridge(alpha = best_alpha, fit_intercept = True)
ridge.fit(X, y)
```

```
Ridge(alpha=10.0)
```

```
X.insert(0, 'const', 1)
pd.Series(np.concatenate(([ridge.intercept_], ridge.coef_)), index = X.columns)
```

```
const       -12.834548
cancervol     2.177029
weight        0.011266
```

```
age          -0.463290
benpros       1.168875
capspen       1.729592
gleason       6.048843
vesinv        9.073683
dtype: float64
```

Lasso regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.metrics import mean_squared_error

# Read data
prostate_cancer = pd.read_csv("prostate_cancer.csv")

dummies = pd.get_dummies(prostate_cancer[['vesinv']])

y = prostate_cancer.psa

# Drop the column with the independent variable (Salary), and columns for which we created du
X_ = prostate_cancer.drop(['psa', 'vesinv', 'subject'], axis=1).astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['vesinv']]], axis=1)

lassocv = LassoCV(alphas = None, cv = 97, max_iter = 100000, fit_intercept=True)
lassocv.fit(X, y)
```

```
LassoCV(cv=97, max_iter=100000)
```

```
best_alpha = lassocv.alpha_

n = 97
test_error = np.zeros(n)
```

```python
for i in range(n):
    train_x = X.drop(i, axis=0)
    train_y = y.drop(i)
    test_x = X.iloc[i, :].values.reshape(1, -1)
    test_y = y.iloc[i]

    lasso_mod = Lasso(alpha=best_alpha)
    lasso_mod.fit(train_x, train_y)
    test_pred = lasso_mod.predict(test_x)
    test_error[i] = (test_pred - test_y)**2
```

```
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
```

```
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
```

```
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
Lasso(alpha=30.175850500729613)
```

```
test_MSE = np.mean(test_error)

lasso = Lasso(max_iter = 10000)
lasso.set_params(alpha = best_alpha, fit_intercept=True)
```

```
Lasso(alpha=30.175850500729613, max_iter=10000)
```

```
lasso.fit(X, y)
```

```
Lasso(alpha=30.175850500729613, max_iter=10000)
```

```
X.insert(0, 'const', 1)
```

```
print(f"The optimal alpha for Lasso regression is: {best_alpha}")
```

```
The optimal alpha for Lasso regression is: 30.175850500729613
```

```
print(f"The test MSE for Lasso regression is: {test_MSE}")
```

```
The test MSE for Lasso regression is: 1161.0103496963725
```

```
# Some of the coefficients are now reduced to exactly zero.
pd.Series(np.concatenate(([lasso.intercept_], lasso.coef_)), index=X.columns)
```

```
const        4.271621
cancervol    2.738824
weight       0.006383
age         -0.000000
benpros      0.000000
capspen      0.000000
gleason      0.000000
vesinv       0.000000
dtype: float64
```

Principal Component Regression (PCR)

In R

```
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)

# Make vesinv qualitative
prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Fit PCR model with LOOCV
pcr.fit <- pcr(psa ~ ., data = prostate_cancer,
               scale = TRUE, center = TRUE,
               validation = "LOO")
#summary(pcr.fit)

# Find the optimal M chosen via LOOCV.
which.min(MSEP(pcr.fit)$val[1, 1,]) #M=2
```

```
2 comps
      3
```

```
# Compute test MSE via LOOCV for M = 2
pcr.fit.optimal <- pcr(psa ~ ., data = prostate_cancer,
                       scale = TRUE, validation = "LOO",
                       ncomp = 2)
MSEP(pcr.fit.optimal)$val[1, 1,]["2 comps"]
```

```
 2 comps
1109.342
```

The optimally M chosen via LOOCV is 2 and the test MSE of the model via LOOCV is 1109.342.

In Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression, PLSSVD
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import LeaveOneOut


df = pd.read_csv('prostate_cancer.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97 entries, 0 to 96
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   subject    97 non-null     int64
```

```
 1   psa        97 non-null     float64
 2   cancervol  97 non-null     float64
 3   weight     97 non-null     float64
 4   age        97 non-null     int64
 5   benpros    97 non-null     float64
 6   vesinv     97 non-null     int64
 7   capspen    97 non-null     float64
 8   gleason    97 non-null     int64
dtypes: float64(5), int64(4)
memory usage: 6.9 KB
```

```
dummies = pd.get_dummies(df[['vesinv']])

y = df.psa

# Drop the column with the independent variable, and columns for which we created dummy varia
X_ = df.drop(['psa', 'subject', 'vesinv'], axis=1).astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['vesinv']]], axis=1)

pca = PCA()
X_reduced = pca.fit_transform(scale(X))

pd.DataFrame(pca.components_.T).loc[:4,:5]
```

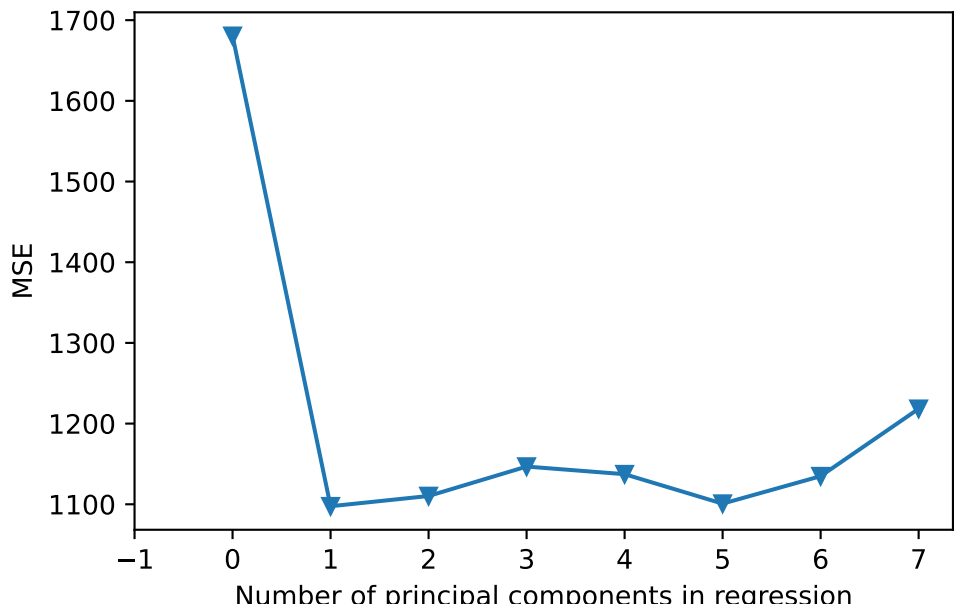
```
          0         1         2         3         4         5
0  0.512816 -0.071634  0.161536 -0.084396 -0.093978  0.672279
1 -0.006585  0.497248  0.764991 -0.001982  0.405291 -0.040558
2  0.117561  0.561815 -0.512764  0.525736  0.303376  0.197867
3 -0.070025  0.645020 -0.057527 -0.314454 -0.683293 -0.057985
4  0.531393 -0.026040  0.124193  0.146989 -0.258481  0.107080
```

```
# LOOCV
n = len(X_reduced)
loo = LeaveOneOut()

regr = LinearRegression()
mse = []

# Calculate MSE with only the intercept (no principal components in regression)
score = -1*model_selection.cross_val_score(regr, np.ones((n,1)), y.ravel(), cv=loo, scoring=
```

```
mse.append(score)

# Calculate MSE using LOOCV for the 7 principle components, adding one component at the time
for i in np.arange(1, 8):
    score = -1*model_selection.cross_val_score(regr, X_reduced[:,:i], y.ravel(), cv=loo, sco
    mse.append(score)

# Plot results
plt.plot(mse, '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.xlim(xmin=-1)
```

```
(-1.0, 7.35)
```

```
plt.show()
```




```
plt.close()

print("Test MSE:", mse)
```

```
Test MSE: [1680.572456652778, 1097.5746772939237, 1110.2650235597341, 1146.7467059779701, 113
```

```
#print("PVE:", np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100))
```

The optimally M chosen via LOOCV is 1 which is different from R(M=2) and the test MSE of the model via LOOCV is 1097.574 which is similar to what we have in R (1109.342). This is probably because sklearn does not have an implementation of PCR like the pls package in R.

Partial Least Squares Regression (PLS)

In R

```
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)

# Make vesinv qualitative
prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Fit PCR model with LOOCV
pls.fit <- plsr(psa ~ ., data = prostate_cancer,
                scale = TRUE, center = TRUE,
                validation = "LOO")
#summary(pls.fit)

# Find the optimal M chosen via LOOCV.
which.min(MSEP(pls.fit)$val[1, 1,]) #M=1
```

```
1 comps
      2
```

```
# Compute test MSE via LOOCV for M = 1
pls.fit.optimal <- plsr(psa ~ ., data = prostate_cancer,
                        scale = TRUE, validation = "LOO",
                        ncomp = 1)
MSEP(pls.fit.optimal)$val[1, 1,]["1 comps"]
```

```
 1 comps
1109.462
```

The optimally M chosen via LOOCV is 1 and the test MSE of the model via LOOCV is 1109.462.

In Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression, PLSSVD
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import LeaveOneOut


df = pd.read_csv('prostate_cancer.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97 entries, 0 to 96
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   subject    97 non-null     int64
 1   psa        97 non-null     float64
 2   cancervol  97 non-null     float64
 3   weight     97 non-null     float64
 4   age        97 non-null     int64
 5   benpros    97 non-null     float64
 6   vesinv     97 non-null     int64
 7   capspen    97 non-null     float64
 8   gleason    97 non-null     int64
dtypes: float64(5), int64(4)
memory usage: 6.9 KB
```

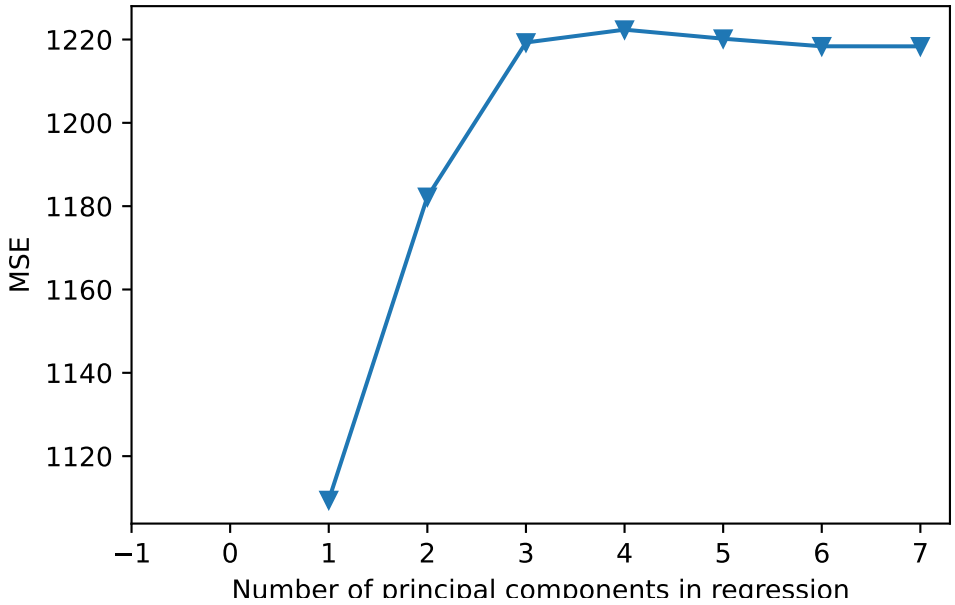
```
dummies = pd.get_dummies(df[['vesinv']])

y = df.psa

# Drop the column with the independent variable, and columns for which we created dummy varia
X_ = df.drop(['psa', 'subject', 'vesinv'], axis=1).astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['vesinv']]], axis=1)

pca = PCA()
X_reduced = pca.fit_transform(scale(X))

pd.DataFrame(pca.components_.T).loc[:4,:5]
```

```
          0         1         2         3         4         5
0  0.512816 -0.071634  0.161536 -0.084396 -0.093978  0.672279
1 -0.006585  0.497248  0.764991 -0.001982  0.405291 -0.040558
2  0.117561  0.561815 -0.512764  0.525736  0.303376  0.197867
3 -0.070025  0.645020 -0.057527 -0.314454 -0.683293 -0.057985
4  0.531393 -0.026040  0.124193  0.146989 -0.258481  0.107080
```

```
# LOOCV
n = len(X_reduced)
loo = LeaveOneOut()

mse = []

# Calculate MSE using LOOCV for the 7 principle components, adding one component at the time
for i in np.arange(1, 8):
    pls = PLSRegression(n_components=i)
    score = model_selection.cross_val_score(pls, scale(X), y,
    cv=loo, scoring='neg_mean_squared_error').mean()
    mse.append(-score)

# Plot results
plt.plot(np.arange(1, 8), np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.xlim(xmin=-1)
```

```
(-1.0, 7.3)
```

```
plt.show()
```




```
plt.close()

print("Test MSE:", mse)
```

```
Test MSE: [1109.4616769059044, 1182.2280708011424, 1219.283292432951, 1222.3639202815616, 12
```

The optimally M chosen via LOOCV is 1 and the test MSE of the model via LOOCV is 1109.462, which are identical to what we have in R.

Principal Component Analysis (PCA)

The `state.x77` dataset available from datasets package in R. The 50×8 data matrix gives statistics on 8 features of the 50 states in the US from 1977.

In R

Exploratory analysis

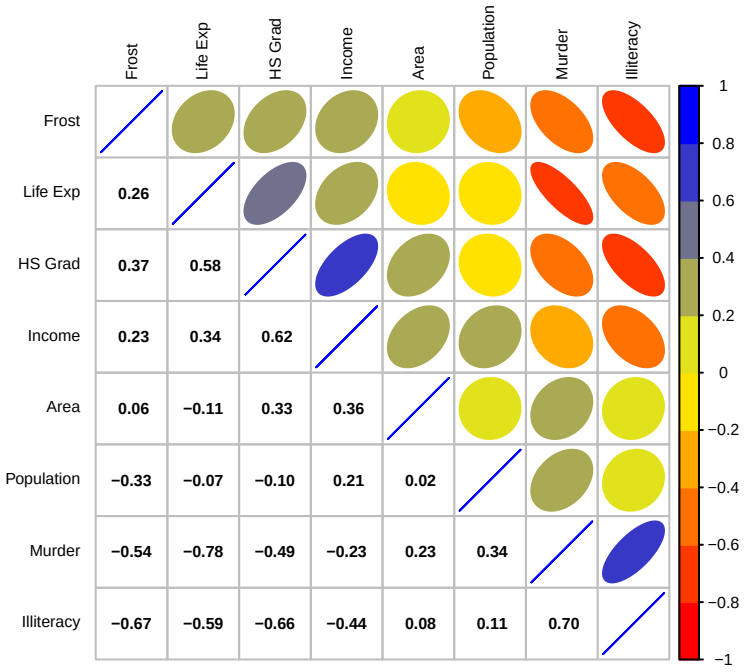
```r
state_data <- datasets::state.x77
# Summary statistics
dim(state.x77)
```

```
[1] 50  8
```

```r
summary(state.x77)
```

```
   Population        Income       Illiteracy       Life Exp
 Min.   :  365   Min.   :3098   Min.   :0.500   Min.   :67.96
 1st Qu.: 1080   1st Qu.:3993   1st Qu.:0.625   1st Qu.:70.12
 Median : 2838   Median :4519   Median :0.950   Median :70.67
 Mean   : 4246   Mean   :4436   Mean   :1.170   Mean   :70.88
 3rd Qu.: 4968   3rd Qu.:4814   3rd Qu.:1.575   3rd Qu.:71.89
 Max.   :21198   Max.   :6315   Max.   :2.800   Max.   :73.60
     Murder          HS Grad          Frost             Area
 Min.   : 1.400   Min.   :37.80   Min.   :  0.00   Min.   :  1049
 1st Qu.: 4.350   1st Qu.:48.05   1st Qu.: 66.25   1st Qu.: 36985
 Median : 6.850   Median :53.25   Median :114.50   Median : 54277
 Mean   : 7.378   Mean   :53.11   Mean   :104.46   Mean   : 70736
 3rd Qu.:10.675   3rd Qu.:59.15   3rd Qu.:139.75   3rd Qu.: 81163
 Max.   :15.100   Max.   :67.30   Max.   :188.00   Max.   :566432
```

```r
# Analyzing the relationship among variables
corMatrix <- cor(as.matrix(state.x77)) # Calculate correlation matrix
col <- colorRampPalette(c("red", "yellow", "blue")) # 3 colors to represent coefficients -1 t
corrplot.mixed(corMatrix, order = "AOE", lower = "number", lower.col = "black",
number.cex = .5, tl.cex = 0.5, cl.cex = 0.5,
upper = "ellipse", upper.col = col(10),
diag = "u", tl.pos = "lt", tl.col = "black")
```



This dataset contains 50 states information in the early years around the 1970s in the USA. The information including `Population`, `Income`: per capita income, `Illiteracy`, `Life Exp`: life expectancy in years, `Murder`: murder and non-negligent manslaughter rate per 100,000 population, `HS Grad`: percent high-school graduates, `Frost`: mean number of days with minimum temperature below freezing in capital or large city, and `Area`: land area in square miles.

By checking the correlation plot of the variables(the plot is attached at the end of the regular report before Bonus), we see a positive correlation of 0.7 between `Murder` and `Illiteracy` which means that the lower education level the state has, the higher chance of murder rate the state will have; Negative correlations (-0.78, -0.54) between `Murder` and `Life.Exp`, `Frost` illustrate that the more occurrence of murder, the shorter life the state will expect; And the colder the weather, the lower chance the murder will occur; a negative correlation of -0.66 between `HS Grad` and `Illiteracy` means that the higher percent of high-school graduates the state has, a lower education level the state will have. The variables in the dataset are linearly correlated.

Standardizing the variables

Yes, standardizing the variables is necessary before performing the analysis especially for PCA since the the variables in the dataset are measured in different units. For example, the `Area` are measured in square miles while `Illiteracy` are described in percent, there is a huge difference in the variance of the two variables (7280748061 for Area and 0.37 for Illiteracy). The optimization in PCA is to maximize the sample variance, if the variance of one variable is higher than others, the PCA components will be biased in that direction.

PCA

```
state_data <- datasets::state.x77

state_data <- as.data.frame(state_data)
rownames(state_data) <- state.abb[match(rownames(state_data), state.name)]

# Standardize the variables and perform PCA
pca <- prcomp(state_data, center = T, scale = T)

# Get the loading matrix
loading_matrix <- pca$rotation

kable(loading_matrix, caption = "The Loading Matrix")
```

Table 5: The Loading Matrix

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Population	0.1264281	0.4108742	-0.6563255	-0.4093856	-0.4059464	0.0106562	0.0621587	0.2192465
Income	-0.2988299	0.5189788	-0.1003592	-0.0884466	0.6375870	-0.4617702	-0.0091047	-0.0602920
Illiteracy	0.4676692	0.0529687	0.0708985	0.3528280	-0.0035260	-0.3874158	0.6198003	0.3386884
Life Exp	-0.4116104	-0.0816561	-0.3599330	0.4425633	-0.3265997	-0.2190816	0.2562131	-0.5274333
Murder	0.4442567	0.3069493	0.1084675	-0.1656002	0.1280687	0.3251961	0.2950432	-0.6782513
HS Grad	-0.4246844	0.2987666	0.0497085	0.2315741	0.0992646	0.6446465	0.3930192	0.3072418
Frost	-0.3574124	-0.1535841	0.3871145	-0.6186512	-0.2173638	-0.2126841	0.4720131	-0.0283444
Area	-0.0333846	0.5876245	0.5103850	0.2011255	-0.4985063	0.1483605	0.2862602	0.0132032

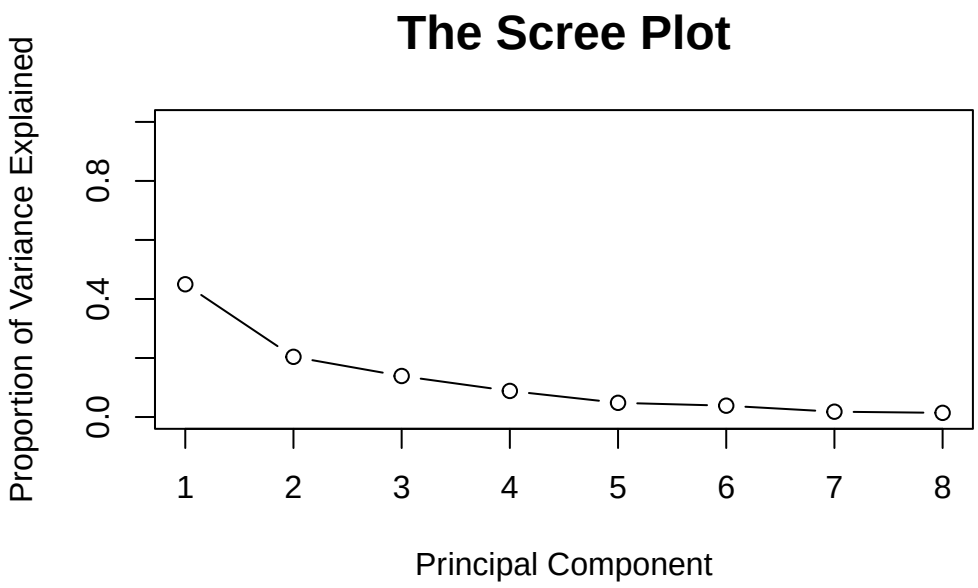
```
# Compute the proportion of variance explained (PVE) by each of the component
pc.var <- pca$sdev^2
pve <- pc.var/sum(pc.var)

pve_df <- t(data.frame(pve))
colnames(pve_df) <- c("PC1", "PC2", "PC3", "PC4", "PC5", "PC6", "PC7", "PC8")
```

```r
rownames(pve_df) <- NULL

kable(pve_df, caption = "The Proportion of Variance Explained By Each of the Principle Compo
```

Table 6: The Proportion of Variance Explained By Each of the Principle Component

PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
0.4498619	0.2039899	0.1389926	0.088438	0.0480802	0.0384327	0.0180561	0.0141485

```r
# Scree plot
plot(pve, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     main = "The Scree Plot",
     ylim = c(0,1),
     type = 'b',
     cex.main = 1.5,
     cex.lab = 1,
     cex.axis = 1)
```



The loading matrix and the Proportion of Variance Explained (PVE) table provide the loadings of each component and the proportion of variance accounted for by each PC. After reviewing

the PVEs and examining the scree plot, I would suggest retaining 5 PCs. This decision is based on the observation that beyond the 5th PC, there is minimal variation in the PVEs. Specifically, the difference between the 5th and 6th PCs is only 0.0096, whereas the difference between the 4th and 5th PCs is notably larger at 0.04. This suggests that the first 5 PCs capture a substantial portion of the variance in the data.

The first two PCs

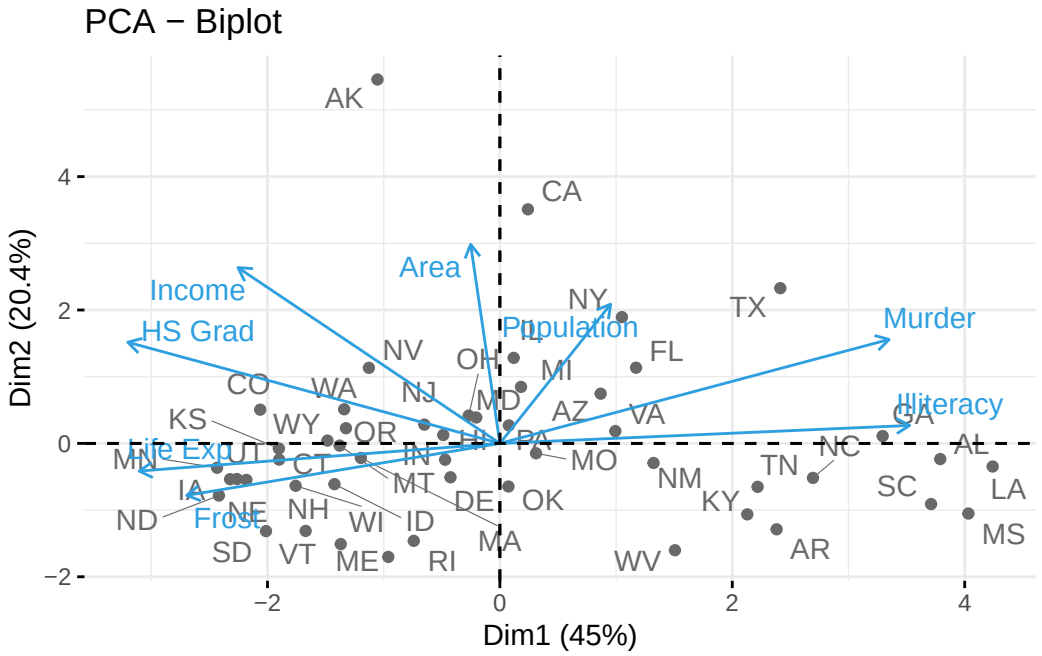
```
# The correlations between PC and standardized variables
loading_matrix_2PC <- pca$rotation[, 1:2]
pc.var <- pca$sdev^2
var_2PC <- pc.var[1:2]

corr_PC1 <- sqrt(var_2PC[1]) * loading_matrix[,1]
corr_PC2 <- sqrt(var_2PC[2]) * loading_matrix[,2]

# The cumulative PVEs
pve <- pc.var/sum(pc.var)
cum_pve_2PC <- cumsum(pve)[1:2]

# Create the table
corr_PC1 <- c(corr_PC1, cum_pve_2PC[1])
corr_PC2 <- c(corr_PC2, cum_pve_2PC[2])
table1_df <- data.frame(corr_PC1, corr_PC2)
colnames(table1_df) <- c("PC1", "PC2")
rownames(table1_df) <- c(rownames(loading_matrix), "Cumulative PVE")
kable(table1_df,
      caption = "The Correlations of the Standardized Variables with the PCs and Cumulative P
```

Table 7: The Correlations of the Standardized Variables with the PCs and Cumulative PVEs of the PCs

	PC1	PC2
Population	0.2398436	0.5248778
Income	-0.5669029	0.6629778
Illiteracy	0.8872037	0.0676657
Life Exp	-0.7808560	-0.1043129
Murder	0.8427885	0.3921173
HS Grad	-0.8056584	0.3816642
Frost	-0.6780384	-0.1961984
Area	-0.0633331	0.7506702

	PC1	PC2
Cumulative PVE	0.4498619	0.6538519

```
# Biplot
fviz_pca_biplot(pca, repel = TRUE,
                col.var = "#2E9FDF", # Variables color
                col.ind = "#696969")  # Individuals color
```



The correlations of the standardized variables with the components were calculated by $corr(Z_k, X_j) = \frac{\sqrt{\lambda_k}\phi_{jk}}{sd(X_j)}$, since the variables were standardized before PCA(with center = T, scale = T), the standard deviation of X_j is 1. Consequently, the correlations simplify to the product of the component's standard deviation with the corresponding loadings. Based on the table, we see that the first PC exhibits strong positive correlations with Illiteracy and Murder while the second PC displays notable positive correlations with Area and Income.

The cumulative PVE for the first two Principal Components (PCs) are 0.4498619 and 0.6538519, respectively. This means that the first two PCs collectively account for 65% of the total variability present in the data.

The biplot reveals distinctive patterns in the loadings of the first two PCs. The first loading vector assigns roughly equal positive significance to Illiteracy and Murder, while assigning similar but negative weights to Life Expectancy, High School Graduation Rate, and Frost.

It assigns comparatively less weight to Population, Income, and Area. The second loading vector allocates the majority of its weight to Area, Income, and Population, while assigning considerably less importance to the remaining three features.

This implies that states with notably high positive scores on the first component, such as Alabama, Louisiana, and Mississippi, tend to exhibit elevated rates of Murder and Illiteracy. Conversely, states like North Dakota and Minnesota, which have negative scores on the first component, tend to have lower rates of Murder and Illiteracy. A similar observation can be made for the second component, where states with high scores, largely driven by Area and Income, exhibit distinctive characteristics in contrast to states with lower scores on this component.

The "southern" component is the first PC since it has higher positive loadings on variables like Murder and Illiteracy. States with positive scores on this component, such as Alabama, Louisiana, and Mississippi, exhibit characteristics that align with common perceptions or historical associations of the southern region of the United States.

Clustering

The planet data stored in the `planet.csv` file consists values of three features for 101 exoplanets discovered up to October 2002. We are interested in clustering the exoplanets on the basis of these features. The features recorded are `Mass` (in Jupiter mass), `Period` (in Earth days), and `Eccentricity`.

In R

Exploratory analysis

```
planet_data <- read.csv("planet.csv")
head(planet_data)
```

```
   Mass Period Eccentricity
1 0.120  4.950         0.00
2 0.197  3.971         0.00
3 0.210 44.280         0.34
4 0.220 75.800         0.28
5 0.230  6.403         0.08
6 0.250  3.024         0.02
```

```
# Correlations
cor_matrix <- cor(planet_data)
kable(cor_matrix, caption = "The Correlation Coefficients")
```

Table 8: The Correlation Coefficients

	Mass	Period	Eccentricity
Mass	1.0000000	0.2684085	0.3049333
Period	0.2684085	1.0000000	0.1445935
Eccentricity	0.3049333	0.1445935	1.0000000

The histograms and scatter plots are shown at end of this section(before Bonus).

Form the histograms, we see that both Mass and Period exhibit a right-skewed distribution. The medians for Mass and Period are 1.76 and 337.110, respectively, which are lower than their respective means of 3.327 and 666.531. On the other hand, the distribution of Eccentricity shows no significant skewness, with a nearly identical median (0.27) and mean (0.2815). Notably, in both the Period and Eccentricity variables, there exists an outlier with a value significantly greater than the majority of the observations.

Form the pairwise scatter plot and the correlation coefficients, we can see that there exists a positive relationship between the variables. This suggests that a higher value in one variable tends to be associated with a higher value in the other variable. However, there is no strong linear relationship between the variables based on the correlation coefficients.

Standardizing the variables

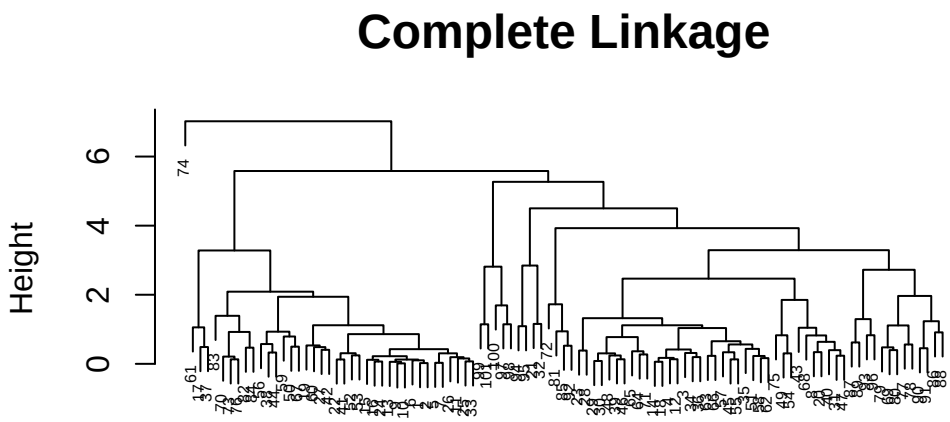
Yes, standardizing the variables before clustering is a good idea since the the variables in the dataset are measured in different units. For example, the `Period` are measured in Earth days while `Mass` are described in Jupiter mass, there is a huge difference in the range of the two variables (5357.01 for Period and 17.45 for Mass). The features having large ranges will implicitly assign greater efforts in the distances compared to the features having smaller ranges. Furthermore, the features need to be dimensionless since the numerical values of the ranges of dimensional features rely upon the units of measurements and, hence, a selection of the units of measurements may significantly alter the outcomes of clustering.

Metric/correlation-based distance

Given the scatter plots in part (a) and the correlation coefficients between the variables (which are relatively small, with values of 0.3, 0.27, and 0.14), I would use a metric-based distance measure. In this context, there isn't a clear linear relationship between the variables. Moreover, the absolute values of the variables hold more significance than the inter-variable relationships. Therefore, utilizing a metric-based distance, such as the Euclidean distance, aligns well with the characteristics of the data.

Hierarchically cluster

```
# Standardize the variables
planet_data_scale <- planet_data %>% mutate_if(is.numeric, scale)

# Hierarchical clustering with complete linkage and Euclidean distance
hc.complete <- hclust(dist(planet_data_scale), method = "complete")

# Plot the dendograms
plot(hc.complete, main = "Complete Linkage", cex = 0.5, xlab = "", sub = "",
     cex.main = 1.5,      # Adjusts text size of main title
     cex.lab = 1,
     cex.axis = 1)
```



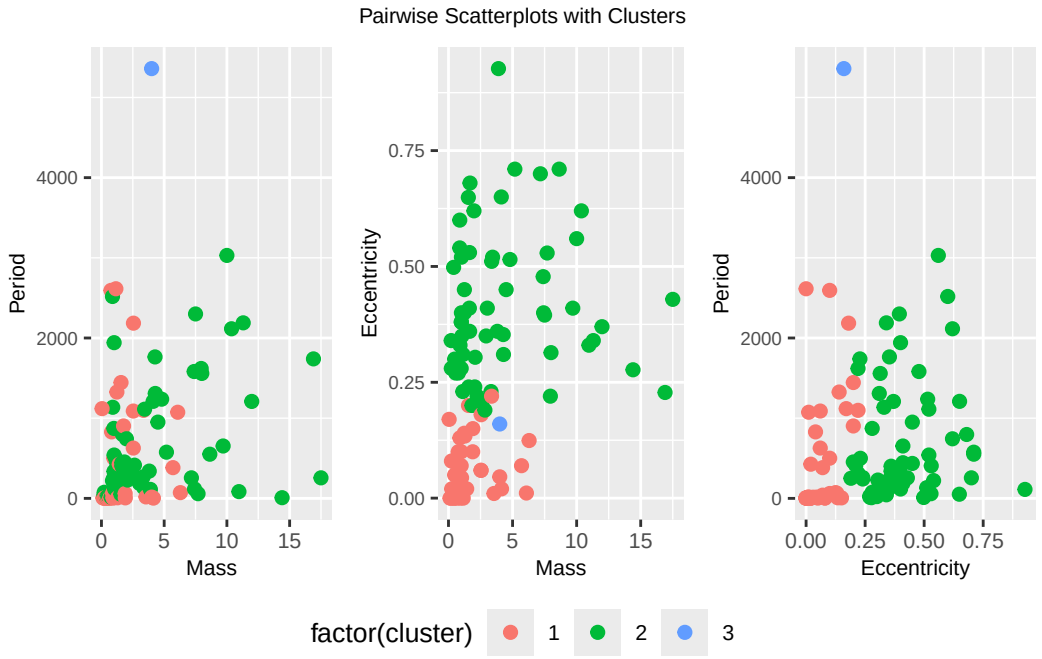

```
# Cut the dendograms to determine cluster labels
labels <- cutree(hc.complete, 3)

# Add the cluster labels to the data
planet_data$cluster <- labels

# Cluster means
means_h <- planet_data %>% group_by(cluster) %>% summarise(means_Mass = mean(Mass),
                                                           means_Period = mean(Period),
```

```
                                                means_Eccentricity = mean(Eccentricity))

kable(means_h, caption = "Cluster Means of the Variables")
```

Table 9: Cluster Means of the Variables

cluster	means_Mass	means_Period	means_Eccentricity
1	1.703316	488.7479	0.0707711
2	4.311774	699.7941	0.4126935
3	4.000000	5360.0000	0.1600000

```
# Individual plots
scatter1 <- ggplot(planet_data, aes(x = Mass, y = Period, color = factor(cluster))) +
  geom_point(size=2)+ theme(text = element_text(size = 10),
        axis.text.y = element_text(size = 7),
        axis.title = element_text(size = 8))

scatter2 <- ggplot(planet_data, aes(x = Mass, y = Eccentricity, color = factor(cluster))) +
  geom_point(size=2) + theme(text = element_text(size = 10),
        axis.text.y = element_text(size = 7),
        axis.title = element_text(size = 8))

scatter3 <- ggplot(planet_data, aes(x = Eccentricity, y = Period, color = factor(cluster))) +
  geom_point(size=2) + theme(text = element_text(size = 10),
        axis.text.y = element_text(size = 7),
        axis.title = element_text(size = 8))

# Combine the plots
plot_scatter <- ggarrange(scatter1, scatter2, scatter3,
                      ncol=3, nrow=1, common.legend = TRUE, legend="bottom")
annotate_figure(plot_scatter,
                top = text_grob("Pairwise Scatterplots with Clusters", size = 8))
```



Form the scatter plots, we see that there is only one observation that was in the third cluster which is the outlier that has a very large value of Period. For the other two clusters,the observations are not clustered well especially for those who have similar values of period, and the scatter plot for Mass versus Period does not exhibit distinct clustering patterns. While the scatter plots of Mass against Eccentricity, as well as Period against Eccentricity, show clearer clusters for cluster 1 and 2. This suggests that Eccentricity plays a pivotal role in segregating the data into distinct clusters. Specifically, observations with lower Eccentricity values tend to fall within cluster 1, while those with higher Eccentricity values are more likely to belong to cluster 2.

K-means

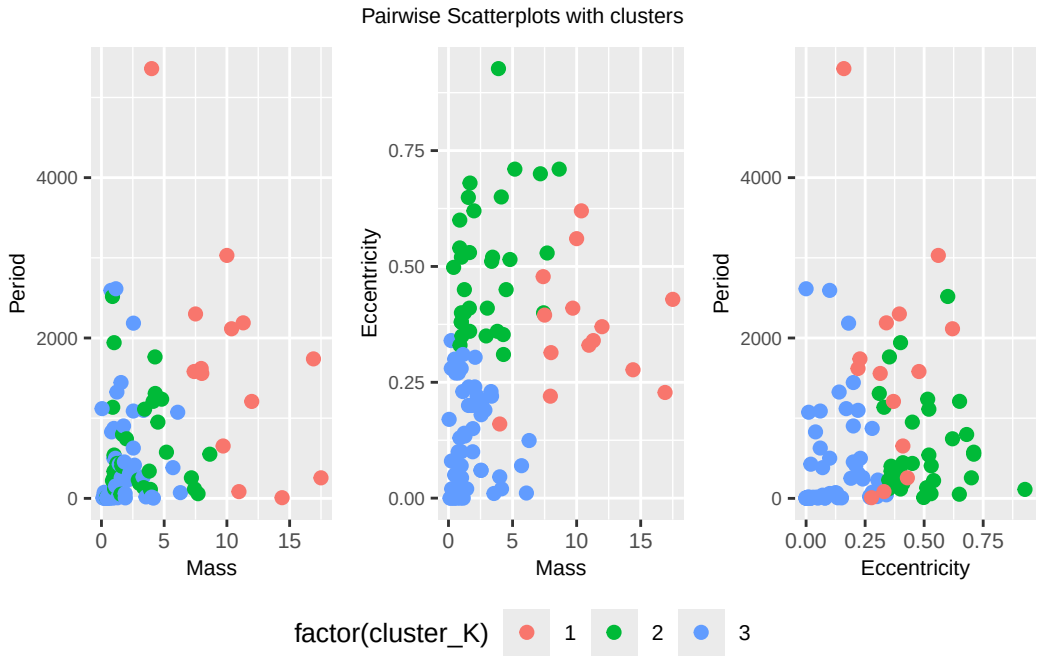
```
km.out <- kmeans(planet_data_scale, centers = 3, nstart = 20)

# Cluster labels
labels_K <- km.out$cluster

# Add the cluster labels to the data
planet_data$cluster_K <- labels_K

# cluster means (original)
means_k <- planet_data %>% group_by(cluster_K) %>% summarise(means_Mass = mean(Mass),
                                                             means_Period = mean(Period),
```

```
                                                    means_Eccentricity = mean(Eccentricity))
kable(means_k, caption = "Cluster Means of the Variables")
```

Table 10: Cluster Means of the Variables

cluster_K	means_Mass	means_Period	means_Eccentricity
1	10.56786	1693.1720	0.3665000
2	3.01000	633.4032	0.5000606
3	1.64400	420.6100	0.1259870

```
# Individual plots
scatter1 <- ggplot(planet_data, aes(x = Mass, y = Period, color = factor(cluster_K))) +
  geom_point(size=2) + theme(text = element_text(size = 10),
        axis.text.y = element_text(size = 7),
        axis.title = element_text(size = 8))

scatter2 <- ggplot(planet_data, aes(x = Mass, y = Eccentricity, color = factor(cluster_K))) +
  geom_point(size=2)+ theme(text = element_text(size = 10),
        axis.text.y = element_text(size = 7),
        axis.title = element_text(size = 8))

scatter3 <- ggplot(planet_data, aes(x = Eccentricity, y = Period, color = factor(cluster_K))) +
  geom_point(size=2) + theme(text = element_text(size = 10),
        axis.text.y = element_text(size = 7),
        axis.title = element_text(size = 8))

# Combine the plots
plot_scatter <- ggarrange(scatter1, scatter2, scatter3,
                      ncol=3, nrow=1, common.legend = TRUE, legend="bottom")
annotate_figure(plot_scatter,
                top = text_grob("Pairwise Scatterplots with clusters", size = 8))
```



Different form the hierarchically cluster result with 3 clusters, the k-means clustering method yields clusters with multiple observations in each group, and designates the outlier(large value of period) to cluster 2. And, form the scatter plots, the observations are not clustered well especially for those those with similar Period values. Consistent with the hierarchical clustering outcome, lower values of Eccentricity are associated with one specific cluster, while higher values of Eccentricity are more prevalent in the other two clusters. And, higher Mass values tend to be concentrated within one cluster, while lower Mass values are more commonly found in the remaining two clusters.

Comparison

The hierarchical clustering approach is recommend, especially with the selection of three clusters. This method effectively isolates the outlier which has an exponentially large Period value, by treating it as a distinct cluster. Additionally, it provides more distinct clusters for the remaining observations, enhancing the overall clarity of the clustering results.

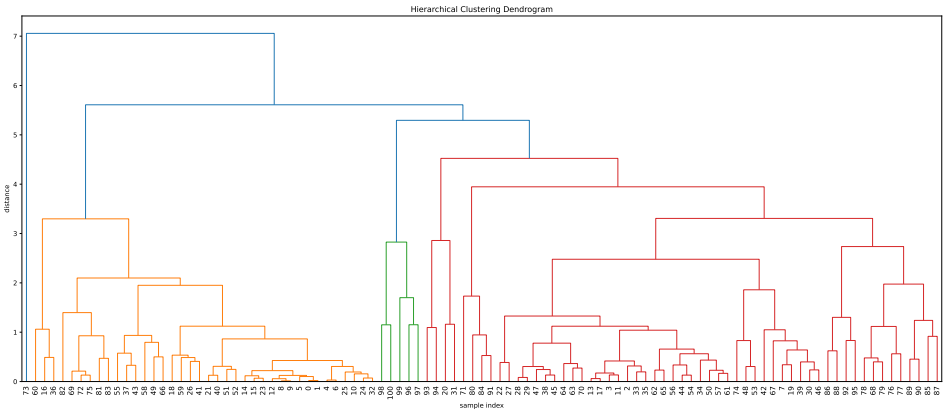
In Python

Hierarchically cluster

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram

# Read data
planet_data = pd.read_csv("planet.csv")

# Select numeric columns and scale them
numeric_columns = planet_data.select_dtypes(include=['number']).columns
scaler = StandardScaler()

planet_data_scale = planet_data.copy()  # Create a copy to avoid modifying the original data

# Scale numeric columns
planet_data_scale[numeric_columns] = scaler.fit_transform(planet_data[numeric_columns])


hc_complete = linkage(planet_data_scale, "complete", metric='euclidean')

# calculate full dendrogram
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    hc_complete,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=11.,  # font size for the x axis labels
)
```

```
{'icoord': [[25.0, 25.0, 35.0, 35.0], [15.0, 15.0, 30.0, 30.0], [65.0, 65.0, 75.0, 75.0], [55
```

```
plt.show()
```




```
# Cut the dendrogram
from scipy.cluster.hierarchy import cut_tree
print(cut_tree(hc_complete, n_clusters = 3).T) # Printing transpose just for space
```

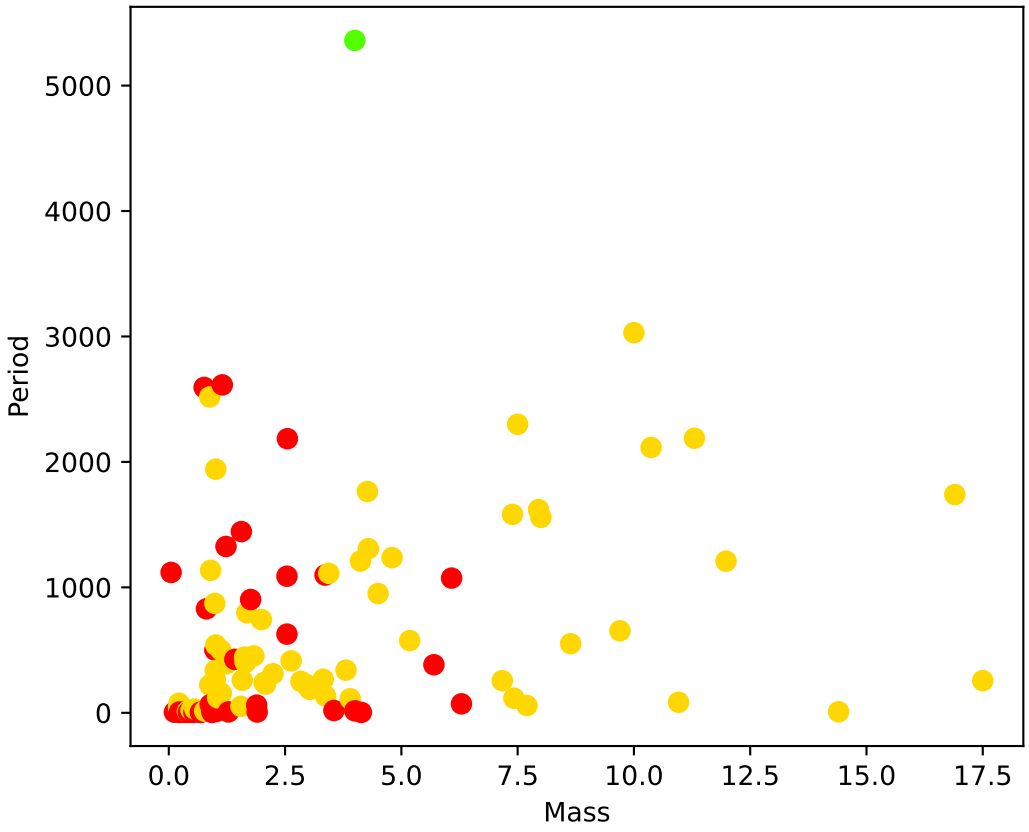
```
[[0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 1 1 1 1 0 1 1 1
  0 0 1 1 0 0 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1 1 0 0 0 1 1 1 1 1 0 1 1 0 1 1
  0 2 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]]
```

```
planet_data['Cluster_H'] = cut_tree(hc_complete, n_clusters = 3)

mean_values = planet_data.groupby('Cluster_H')[['Mass', 'Period', 'Eccentricity']].mean()

print("Cluster means in Hierarchically cluster:", mean_values)
```

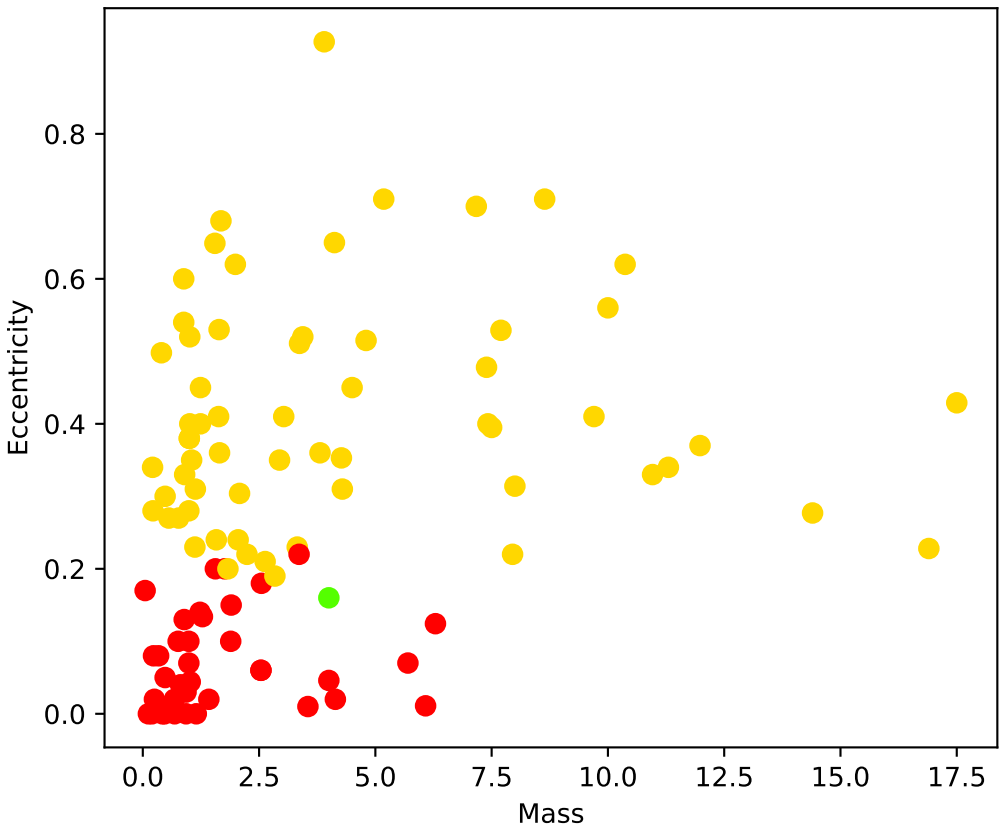
```
Cluster means in Hierarchically cluster:                Mass       Period  Eccentricity
Cluster_H
0           1.703316   488.747927      0.070771
1           4.311774   699.794116      0.412694
2           4.000000  5360.000000      0.160000
```

```
plt.figure(figsize=(6,5))
plt.scatter(planet_data.iloc[:,0], planet_data.iloc[:,1], s=50,
c=planet_data['Cluster_H'], cmap=plt.cm.prism)
plt.xlabel('Mass')
```

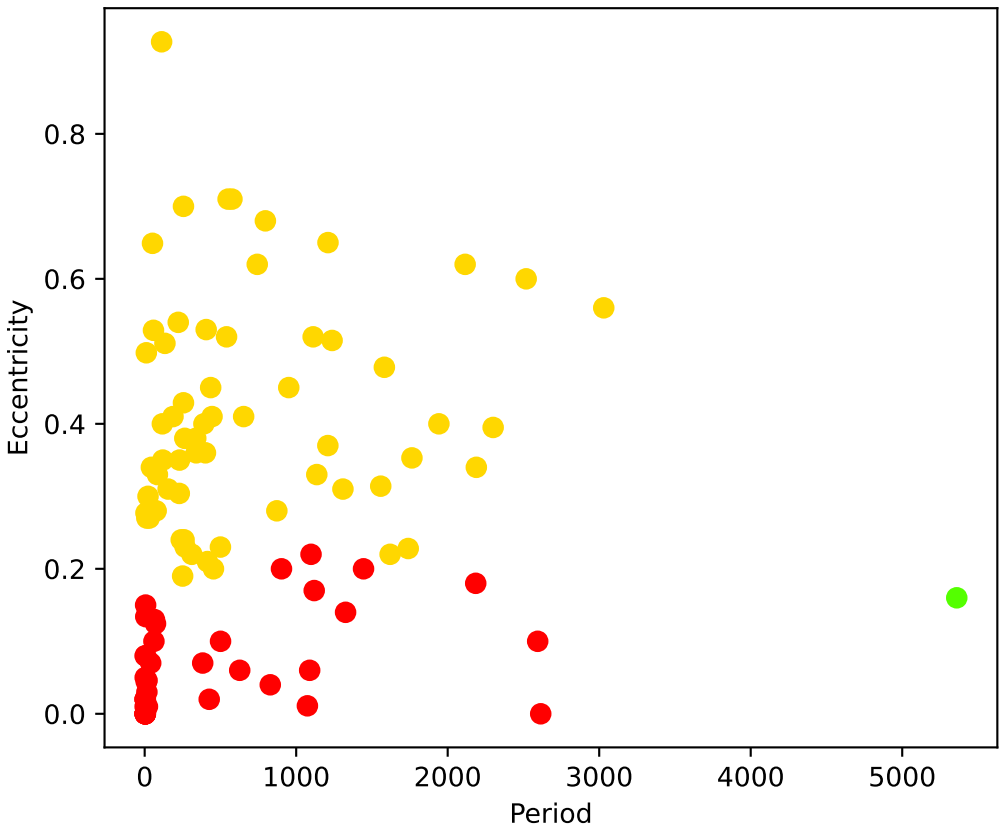
```
plt.ylabel('Period')
plt.show()
```




```
plt.close()

plt.figure(figsize=(6,5))
plt.scatter(planet_data.iloc[:,0], planet_data.iloc[:,2], s=50,
c=planet_data['Cluster_H'], cmap=plt.cm.prism)
plt.xlabel('Mass')
plt.ylabel('Eccentricity')
plt.show()
```




```
plt.close()

plt.figure(figsize=(6,5))
plt.scatter(planet_data.iloc[:,1], planet_data.iloc[:,2], s=50,
c=planet_data['Cluster_H'], cmap=plt.cm.prism)
plt.xlabel('Period')
plt.ylabel('Eccentricity')
plt.show()
```




```
plt.close()
```

K-means

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Read data
planet_data = pd.read_csv("planet.csv")

# Select numeric columns and scale them
```

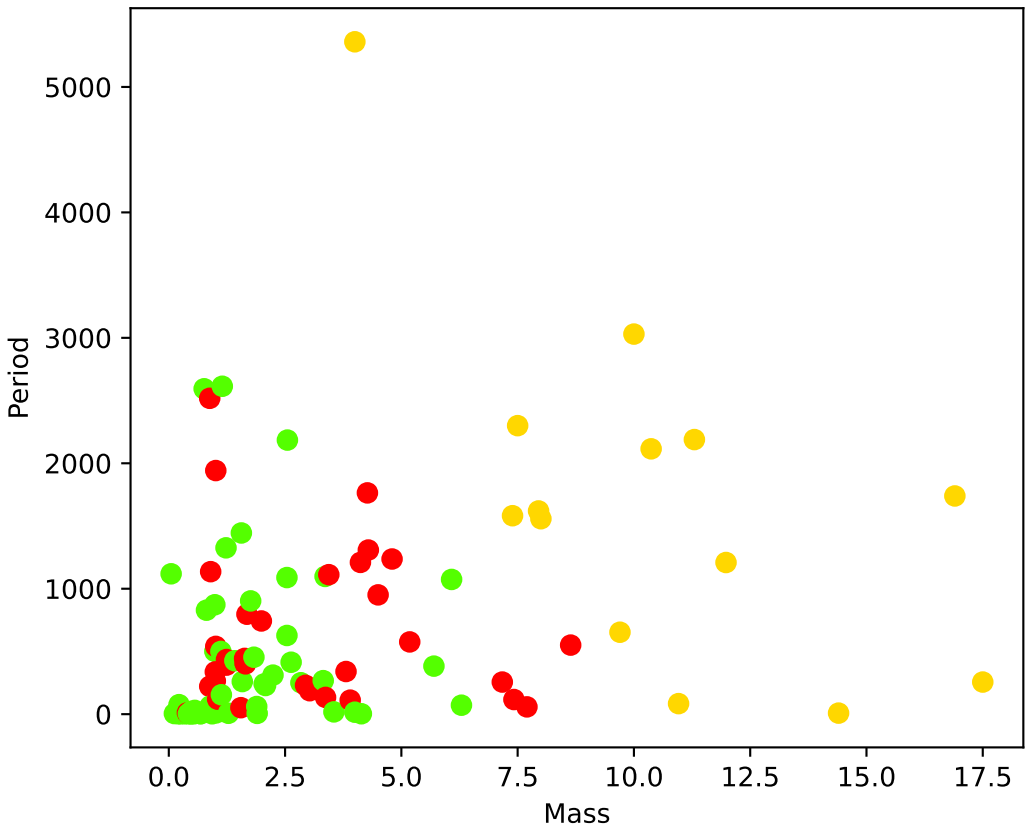
```
numeric_columns = planet_data.select_dtypes(include=['number']).columns
scaler = StandardScaler()

planet_data_scale = planet_data.copy()  # Create a copy to avoid modifying the original data

# Scale numeric columns
planet_data_scale[numeric_columns] = scaler.fit_transform(planet_data[numeric_columns])


kmeans_3_clusters = KMeans(n_clusters = 3, random_state = 123)
kmeans_3_clusters.fit(planet_data_scale)
```

```
KMeans(n_clusters=3, random_state=123)
```

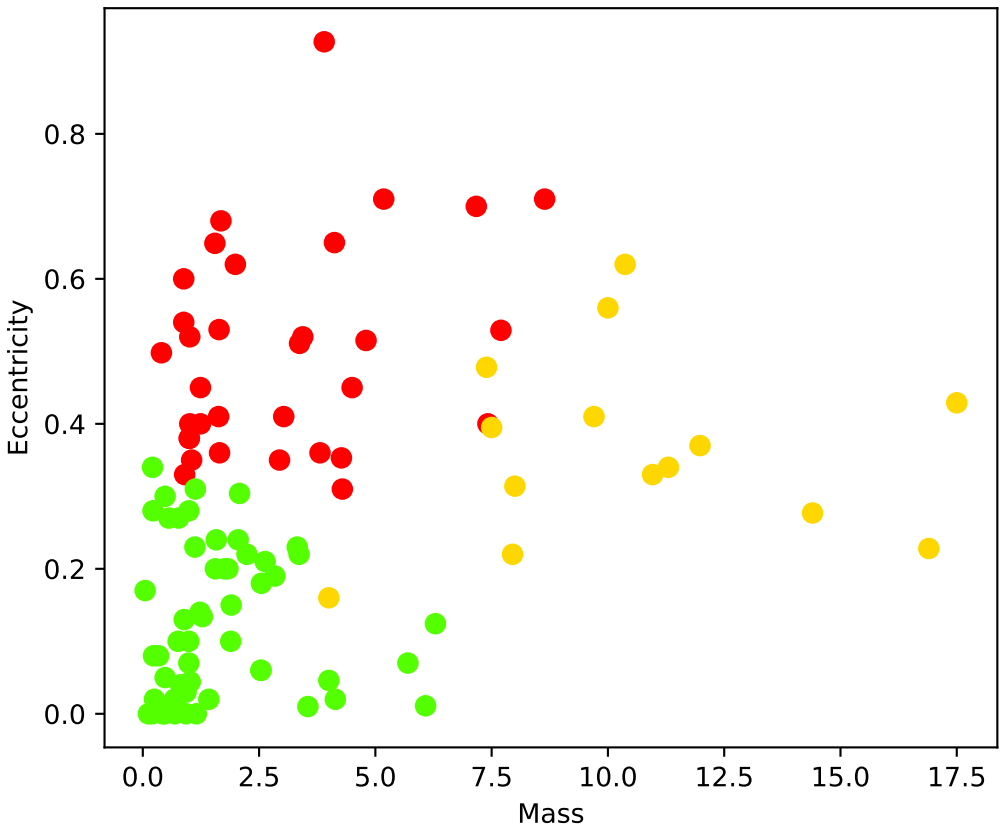
```
planet_data['Cluster_k'] = kmeans_3_clusters.labels_

mean_values = planet_data.groupby('Cluster_k')[['Mass', 'Period', 'Eccentricity']].mean()

print("Cluster means in k-means:", mean_values)
```

```
Cluster means in k-means:                Mass       Period  Eccentricity
Cluster_k
0           3.010000   633.403242      0.500061
1          10.567857  1693.172014      0.366500
2           1.644000   420.610023      0.125987
```

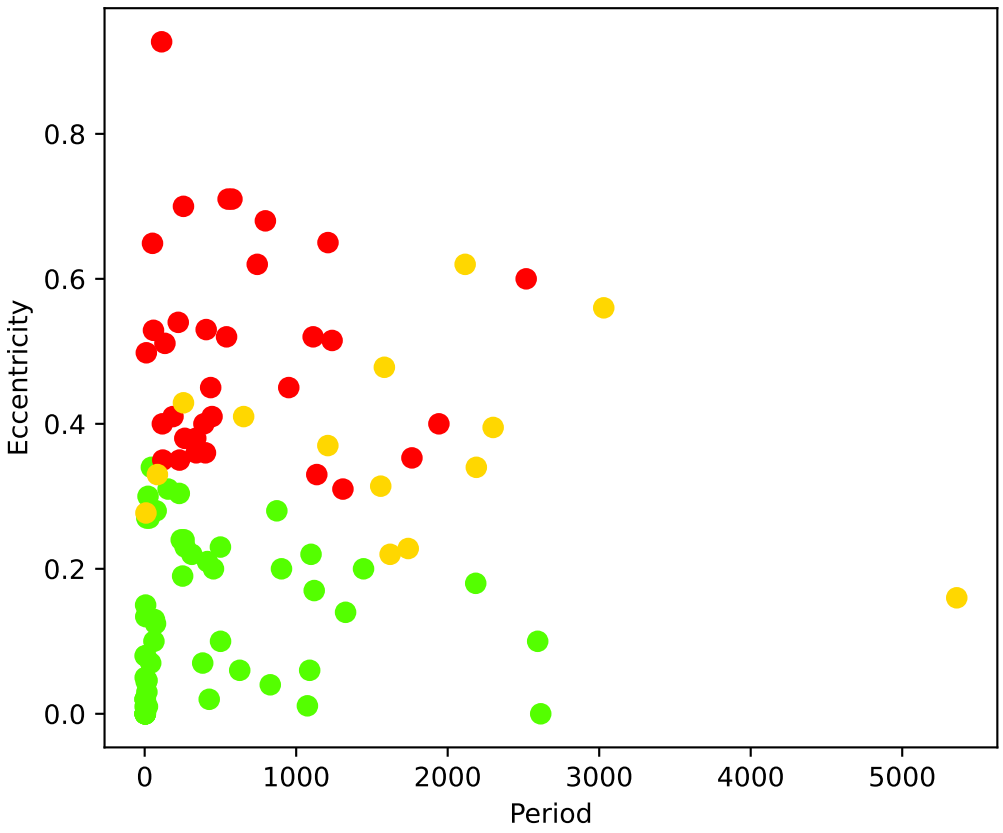
```
plt.figure(figsize=(6,5))
plt.scatter(planet_data.iloc[:,0], planet_data.iloc[:,1], s=50,
c=planet_data['Cluster_k'], cmap=plt.cm.prism)
plt.xlabel('Mass')
plt.ylabel('Period')
plt.show()
```




```
plt.close()

plt.figure(figsize=(6,5))
plt.scatter(planet_data.iloc[:,0], planet_data.iloc[:,2], s=50,
c=planet_data['Cluster_k'], cmap=plt.cm.prism)
plt.xlabel('Mass')
plt.ylabel('Eccentricity')
plt.show()
```




```
plt.close()

plt.figure(figsize=(6,5))
plt.scatter(planet_data.iloc[:,1], planet_data.iloc[:,2], s=50,
c=planet_data['Cluster_k'], cmap=plt.cm.prism)
plt.xlabel('Period')
plt.ylabel('Eccentricity')
plt.show()
```




```
plt.close()
```

Decision Tree

The prostate cancer dataset has psa as the quantitative response variable. Among the predictors, vesinv is a qualitative variable and treat gleason as a quantitative variable. Take all the data as training data. For all the models below, use leave-one-out cross-validation (LOOCV) to compute the estimated test MSE.

In R

A Single Tree

```r
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)

prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Grow a tree
tree.cancer <- tree(psa ~ ., prostate_cancer)
summary(tree.cancer)
```

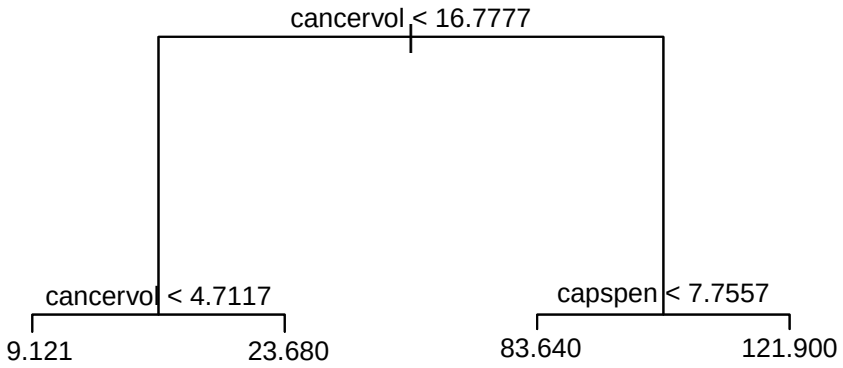
```
Regression tree:
tree(formula = psa ~ ., data = prostate_cancer)
Variables actually used in tree construction:
[1] "cancervol" "capspen"
Number of terminal nodes:  4
Residual mean deviance:  881.9 = 82020 / 93
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-93.680  -6.961  -3.001   0.000   7.160 156.200
```

```r
# LOOCV test MSE
y_preds <- c()
for (i in 1:nrow(prostate_cancer)){
  train.data <- prostate_cancer[-i, ]
  test.data <- prostate_cancer[i, ]
  train.tree <- tree(psa ~ ., train.data)
  y.pred <- predict(train.tree, newdata = test.data)
  y_preds[i] <- y.pred
}
test_mse_a <- mean((y_preds - prostate_cancer$psa)^2)

test_mse_a
```

```
[1] 1438.532
```

```r
# Read data
prostate_cancer <- read.csv("prostate_cancer.csv")

prostate_cancer <- prostate_cancer %>% dplyr::select(-subject)
```

```r
prostate_cancer$vesinv <- as.factor(prostate_cancer$vesinv)

# Grow a tree
tree.cancer <- tree(psa ~ ., prostate_cancer)

# Plot the tree
plot(tree.cancer)
text(tree.cancer, pretty = 0,  cex = 0.8)
```



The constructed regression tree model for predicting 'psa' has 4 terminal nodes. 'cancervol' and 'capspen' were the variables used to split the nodes during the tree construction. The tree's residual mean deviance is 881.9 across 93 data points. The residuals has a range from -93.680 to 156.200, indicating the spread of errors in prediction. Also, while the median residual stands at -3.001, the mean residual is reported as 0.000, signifying the balance between positive and negative errors.

The regions are:

$R_1 = \{X|\text{cancervol} < 4.7117\}$, $R_2 = \{X|16.7777 > \text{cancervol} \geq 4.7117\}$, $R_3 = \{X|\text{cancervol} \geq 16.7777, \text{capspen} < 7.7557\}$, and $R_4 = \{X|\text{cancervol} \geq 16.7777, \text{capspen} \geq 7.7557\}$

The test MSE obtained via LOOCV is 1438.5315415.

LOOCV pruning

```
# Perform cost complexity pruning by CV
set.seed(3)
cv.cancer <- cv.tree(tree.cancer)
cv.cancer
```

```
$size
[1] 4 3 2 1

$dev
[1] 124111.8 133344.4 133488.0 174240.5

$k
[1]      -Inf  3659.033  4344.157 69652.752

$method
[1] "deviance"

attr(,"class")
[1] "prune"         "tree.sequence"
```

```
# Get the best size
cv.cancer$size[which.min(cv.cancer$dev)]
```

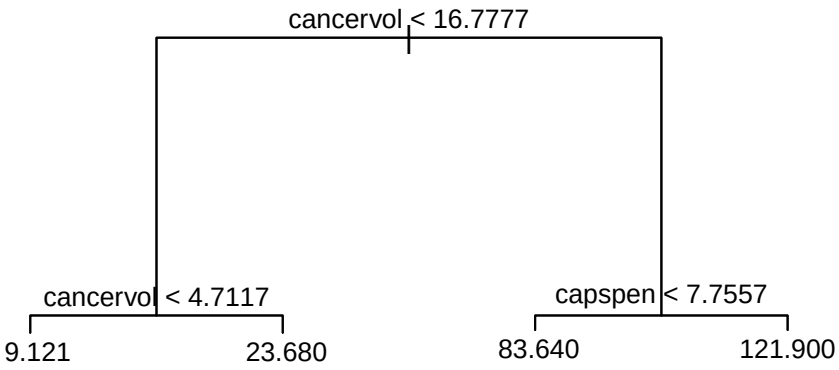
```
[1] 4
```

```
# Get the pruned tree of the best size
prune.cancer <- prune.tree(tree.cancer, best = 4)

# LOOCV test mse for the best pruned tree
y_preds <- c()
for (i in 1:nrow(prostate_cancer)){
  train.data <- prostate_cancer[-i, ]
  test.data <- prostate_cancer[i, ]
  train.tree <- tree(psa ~ ., train.data)
  prune.train.tree <- prune.tree(train.tree, best = 4)
  y.pred <- predict(prune.train.tree, newdata = test.data)
  y_preds[i] <- y.pred
}
test_mse_b <- mean((y_preds - prostate_cancer$psa)^2)
```

```
test_mse_b
```

```
[1] 1434.333
```

```
# Grow a tree
tree.cancer <- tree(psa ~ ., prostate_cancer)

# Get the pruned tree of the best size
prune.cancer <- prune.tree(tree.cancer, best = 4)
plot(prune.cancer)
text(prune.cancer, pretty = 0, cex = 0.8)
```



The best size which has the smallest error rate is 4, which is the same as the original tree, thus, pruning is not helpful in this case. The best pruned and un-pruned trees are the same. The loocv test MSE of the best pruned tree is 1434.3325397. The most important predictors are "cancervol" and "capspen".

Bagging

```r
set.seed(1)

# Bagging, set m = p = 7
bag.cancer <- randomForest(psa ~ ., data = prostate_cancer,
                           mtry = 7, ntree = 1000, importance = TRUE)

# Get variable importance measure for each predictor
importance(bag.cancer)
```

```
              %IncMSE IncNodePurity
cancervol 28.97943655     72300.742
weight     4.91261419      4217.083
age        0.09587962      8321.520
benpros   12.96137277     27454.964
vesinv     5.49994308      2905.976
capspen    0.39752016     33862.787
gleason   -0.79545196       983.411
```

```r
# LOOCV test mse for the best pruned tree
y_preds <- c()
for (i in 1:nrow(prostate_cancer)){
  train.data <- prostate_cancer[-i, ]
  test.data <- prostate_cancer[i, ]
  bag.train <- randomForest(psa ~ ., data = train.data,
                            mtry = 7, ntree = 1000, importance = TRUE)
  y.pred <- predict(bag.train, newdata = test.data)
  y_preds[i] <- y.pred
}
test_mse_c <- mean((y_preds - prostate_cancer$psa)^2)

test_mse_c
```

```
[1] 1274.944
```

The loocv test MSE of is 1274.9441905. The two most important predictors are "cancervol" and "capspen".

Random forest

```r
set.seed(1)

# Bagging, set m = p = 7
rf.cancer <- randomForest(psa ~ ., data = prostate_cancer,
                          mtry = floor(7/3), ntree = 1000, importance = TRUE)

# Get variable importance measure for each predictor
importance(rf.cancer)
```

```
             %IncMSE IncNodePurity
cancervol 12.2990760     45202.871
weight    -2.4621927      9954.881
age       -0.3732772      9943.971
benpros   -0.1335900     10197.325
vesinv     6.4063750     15256.399
capspen   -0.6348785     33063.190
gleason    4.8905419     12521.796
```

```r
# LOOCV test mse for the best pruned tree
y_preds <- c()
for (i in 1:nrow(prostate_cancer)){
  train.data <- prostate_cancer[-i, ]
  test.data <- prostate_cancer[i, ]
  rf.train <- randomForest(psa ~ ., data = train.data,
                           mtry = floor(7/3), ntree = 1000, importance = TRUE)
  y.pred <- predict(rf.train, newdata = test.data)
  y_preds[i] <- y.pred
}
test_mse_d <- mean((y_preds - prostate_cancer$psa)^2)

test_mse_d
```

```
[1] 1164.136
```

The loocv test MSE of is 1164.1360313. The two most important predictors are "cancervol" and "capspen".

Boosting

```r
# Fit a boosted regression tree
set.seed(1)
boost.cancer <- gbm(psa ~ ., data = prostate_cancer, distribution = "gaussian",
                    n.trees = 1000, interaction.depth = 1, shrinkage = 0.01)

# Extracting relative influence scores
relative.influence(boost.cancer, n.trees = 1000, scale. = FALSE, sort. = TRUE)
```

```
cancervol    vesinv    weight   gleason   capspen   benpros       age
1591168.0  867085.3  632908.1  632852.8  387160.9  355911.7  312714.3
```

```r
# LOOCV test mse for the best pruned tree
y_preds <- c()
for (i in 1:nrow(prostate_cancer)){
  train.data <- prostate_cancer[-i, ]
  test.data <- prostate_cancer[i, ]
  boost.train <- gbm(psa ~ ., data = train.data, distribution = "gaussian",
                     n.trees = 1000, interaction.depth = 1, shrinkage = 0.01,
                     verbose = F)
  y.pred <- predict(boost.train, newdata = test.data, n.trees = 1000)
  y_preds[i] <- y.pred
}
test_mse_e <- mean((y_preds - prostate_cancer$psa)^2)

test_mse_e
```

```
[1] 1174.543
```

The loocv test MSE of is 1174.5431272. The two most important predictors are "cancervol" and "vesinv".

Comparison

```r
test_mse <- c(test_mse_a, test_mse_b, test_mse_c,
              test_mse_d, test_mse_e, 1084.374)

results_df <- t(data.frame(test_mse))
colnames(results_df) <- c("Un-prunedTree", "BestPrunedTree", "Bagging",
                          "RandomForest", "Boosting", "Project5")
rownames(results_df) <- NULL
kable(results_df, caption = "LOOCV Test MSE")
```

Table 11: LOOCV Test MSE

Un-prunedTree	BestPrunedTree	Bagging	RandomForest	Boosting	Project5
1438.532	1434.333	1274.944	1164.136	1174.543	1084.374

The random forest method is optimal as it has the smallest LOOCV test MSE among all the tree based models.