CSP571 Data Preparation and Analysis Assignment 2

Poshan Pandey

06/14/2024

2 Practicum Problems

Chapter 4

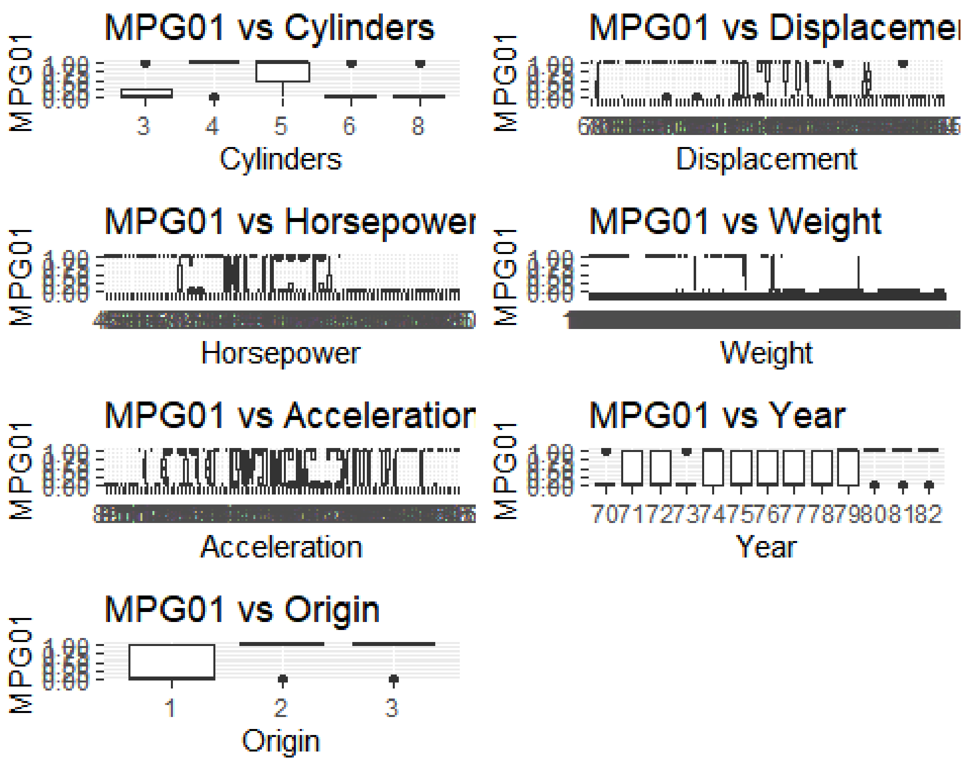
Problem 14

a

```
# Load the dataset
data(Auto, package="ISLR")

# Create mpg01
mpg01 <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)

# Add mpg01 to the Auto data frame
Auto <- data.frame(Auto, mpg01)
```

b

```
# Load necessary libraries
library(ggplot2)
library(gridExtra)

# Create individual boxplots
p1 <- ggplot(Auto, aes(x=factor(cylinders), y=mpg01)) + geom_boxplot() +
labs(title="MPG01 vs Cylinders", x="Cylinders", y="MPG01")
p2 <- ggplot(Auto, aes(x=factor(displacement), y=mpg01)) + geom_boxplot() +
labs(title="MPG01 vs Displacement", x="Displacement", y="MPG01")
p3 <- ggplot(Auto, aes(x=factor(horsepower), y=mpg01)) + geom_boxplot() +
labs(title="MPG01 vs Horsepower", x="Horsepower", y="MPG01")
p4 <- ggplot(Auto, aes(x=factor(weight), y=mpg01)) + geom_boxplot() +
labs(title="MPG01 vs Weight", x="Weight", y="MPG01")
p5 <- ggplot(Auto, aes(x=factor(acceleration), y=mpg01)) + geom_boxplot() +
labs(title="MPG01 vs Acceleration", x="Acceleration", y="MPG01")
p6 <- ggplot(Auto, aes(x=factor(year), y=mpg01)) + geom_boxplot() +
labs(title="MPG01 vs Year", x="Year", y="MPG01")
p7 <- ggplot(Auto, aes(x=factor(origin), y=mpg01)) + geom_boxplot() +
labs(title="MPG01 vs Origin", x="Origin", y="MPG01")

# Arrange the boxplots in a grid
grid.arrange(p1, p2, p3, p4, p5, p6, p7, ncol=2)
```



c

```
# Set seed for reproducibility
set.seed(123)

# Split data into training and test sets
train_indices <- sample(1:nrow(Auto), size = 0.7 * nrow(Auto))
train_set <- Auto[train_indices, ]
test_set <- Auto[-train_indices, ]
```

d

```
# Load necessary library
library(MASS)


# Identify most associated features using t-test
t_test_results <- sort(sapply(1:7, function(i) {
  setNames(abs(t.test(Auto[, i] ~ Auto$mpg01)$statistic), colnames(Auto)[i])
}))
print(t_test_results)

## acceleration         year   horsepower displacement       weight
cylinders
##     7.302430     9.403221    17.681939    22.632004    22.932777
23.035328
##          mpg
##    30.199801
```

```r
# Features most associated with mpg01
# In this example, the top features are identified as:
# cylinders, weight, displacement
most_associated_features <- names(t_test_results)[order(t_test_results,
decreasing = TRUE)[1:3]]
print(most_associated_features)

## [1] "mpg"       "cylinders" "weight"

# Perform LDA on the training data
lda_fit <- lda(mpg01 ~ cylinders + weight + displacement, data = train_set)

# Predict on the test set
lda_pred <- predict(lda_fit, test_set)$class

# Calculate test error
lda_test_error <- mean(lda_pred != test_set$mpg01)
lda_test_error

## [1] 0.1186441
```

e

```r
# Perform QDA
qda_model <- qda(mpg01 ~ cylinders + displacement + horsepower + weight +
acceleration + year + origin, data=train_set)
qda_pred <- predict(qda_model, test_set)$class

# Calculate test error
qda_error <- mean(qda_pred != test_set$mpg01)
qda_error

## [1] 0.06779661
```

f

```r
# Perform logistic regression
logistic_model <- glm(mpg01 ~ cylinders + displacement + horsepower + weight
+ acceleration + year + origin, data=train_set, family=binomial)
logistic_prob <- predict(logistic_model, test_set, type="response")
logistic_pred <- ifelse(logistic_prob > 0.5, 1, 0)

# Calculate test error
logistic_error <- mean(logistic_pred != test_set$mpg01)
logistic_error

## [1] 0.09322034
```

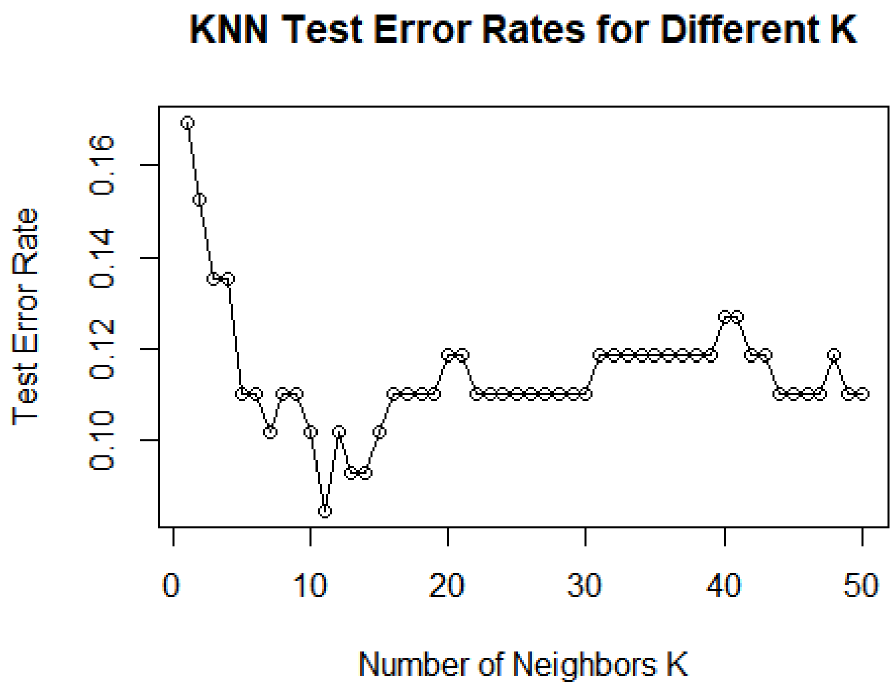
g

```r
# Load necessary library
library(e1071)

# Perform naive Bayes
```

```r
nb_model <- naiveBayes(mpg01 ~ cylinders + displacement + horsepower + weight
+ acceleration + year + origin, data=train_set)
nb_pred <- predict(nb_model, test_set)

# Calculate test error
nb_error <- mean(nb_pred != test_set$mpg01)
nb_error

## [1] 0.09322034
```

h

```r
library(class)
# Prepare data for KNN
train_X <- train_set[, c("cylinders", "weight", "displacement")]
train_y <- train_set$mpg01
test_X <- test_set[, c("cylinders", "weight", "displacement")]

# Perform KNN with different values of K
k_values <- 1:50
knn_errors <- sapply(k_values, function(k) {
  knn_pred <- knn(train_X, test_X, train_y, k = k)
  mean(knn_pred != test_set$mpg01)
})

# Assign names to the errors for plotting
names(knn_errors) <- k_values

# Plot the errors for different K values
plot(k_values, knn_errors, type = "o", xlab = "Number of Neighbors K", ylab =
"Test Error Rate", main = "KNN Test Error Rates for Different K")
```




```r
# Find the best K
best_k <- k_values[which.min(knn_errors)]
best_k

## [1] 11

best_k

## [1] 11
```

11

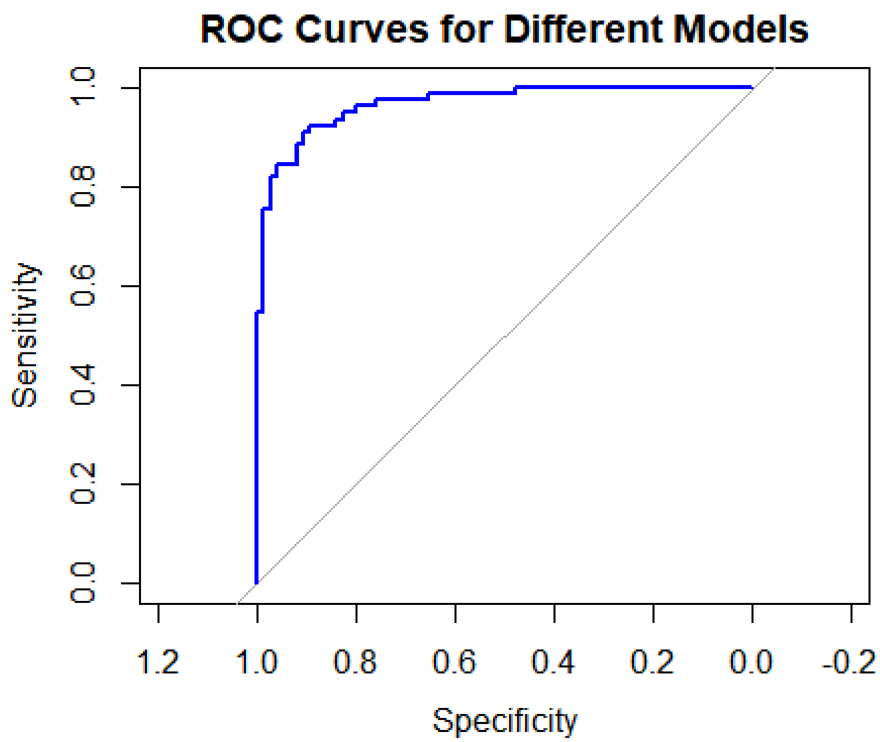
Problem 16

Load and Prepare Data:

```r
# Load the Boston dataset
data(Boston, package = "MASS")

# Create a binary response variable based on the median crime rate
crime_median <- median(Boston$crim)
Boston$crime01 <- ifelse(Boston$crim > crime_median, 1, 0)

# Split the data into a training set and a test set
set.seed(123)  # For reproducibility
train_indices <- sample(1:nrow(Boston), size = 0.7 * nrow(Boston))
train_set <- Boston[train_indices, ]
test_set <- Boston[-train_indices, ]
```

Logistic Regression:

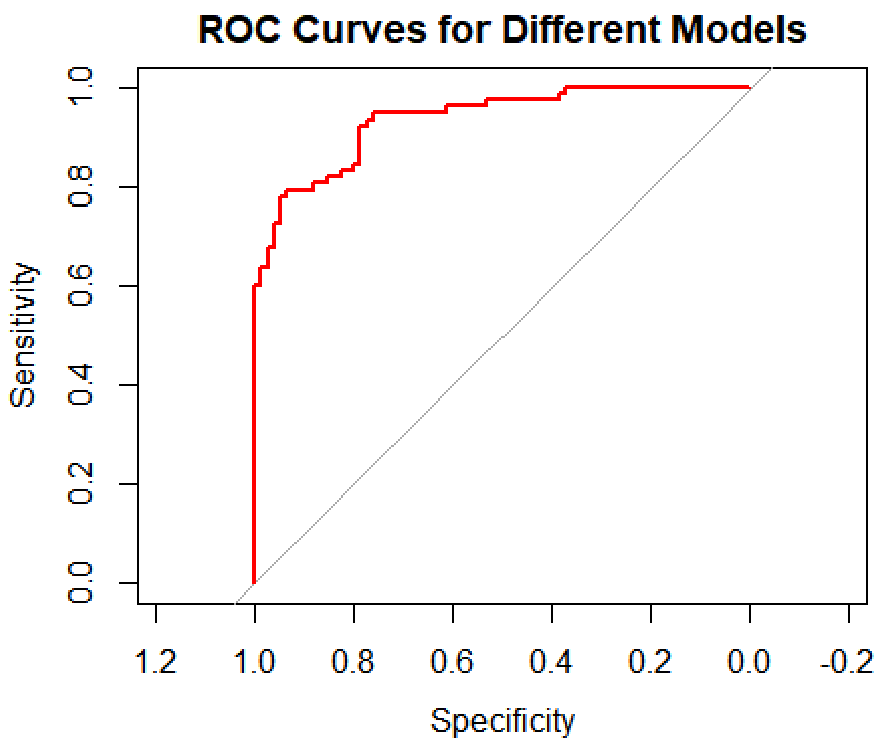
```r
library(pROC)

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

# Perform logistic regression
logistic_model <- glm(crime01 ~ . - crim - crime01, data = train_set, family
= binomial)
logistic_prob <- predict(logistic_model, test_set, type = "response")
logistic_pred <- ifelse(logistic_prob > 0.5, 1, 0)

# Calculate test error
logistic_error <- mean(logistic_pred != test_set$crime01)
logistic_error

## [1] 0.1118421

# ROC curve
logistic_roc <- roc(test_set$crime01, logistic_prob)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

plot(logistic_roc, main = "ROC Curves for Different Models", col = "blue")
```




```r
# Calculate test error
logistic_error <- mean(logistic_pred != test_set$crime01)
logistic_error

## [1] 0.1118421
```

Linear Discriminant Analysis (LDA):

```r
# Perform LDA
lda_model <- lda(crime01 ~ . - crim - crime01, data = train_set)
lda_pred <- predict(lda_model, test_set)$class
lda_prob <- predict(lda_model, test_set)$posterior[,2]

# Calculate test error
lda_error <- mean(lda_pred != test_set$crime01)
print(lda_error)

## [1] 0.1644737

# ROC curve
lda_roc <- roc(test_set$crime01, lda_prob)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

# Ensure to call plot first
plot(lda_roc, main = "ROC Curves for Different Models", col = "red", lwd = 2)
```




```
lda_error

## [1] 0.1644737
```

Naive Bayes:

```
# Perform naive Bayes
nb_model <- naiveBayes(crime01 ~ . - crim - crime01, data = train_set)
nb_pred <- predict(nb_model, test_set)
nb_prob <- predict(nb_model, test_set, type = "raw")[,2]

# Calculate test error
nb_error <- mean(nb_pred != test_set$crime01)
print(nb_error)

## [1] 0.1842105

print(nb_error)

## [1] 0.1842105
```

K-Nearest Neighbors (KNN):

```
# Prepare data for KNN
train_X <- train_set[, !names(train_set) %in% c("crim", "crime01")]
train_y <- train_set$crime01
test_X <- test_set[, !names(test_set) %in% c("crim", "crime01")]

# Perform KNN with k = 5
```

```r
knn_pred <- knn(train_X, test_X, train_y, k = 5)

# Calculate test error
knn_error <- mean(knn_pred != test_set$crime01)
print(knn_error)

## [1] 0.07236842
```

Chapter 5

Problem 5

a

```r
# Load necessary libraries
library(ISLR)

##
## Attaching package: 'ISLR'

## The following object is masked _by_ '.GlobalEnv':
##
##     Auto

library(MASS)

# Load the Default dataset
data(Default)
# Fit the logistic regression model
set.seed(1)  # Setting random seed
logistic_model <- glm(default ~ income + balance, data = Default, family =
binomial)
summary(logistic_model)

##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
```

```
##
## Number of Fisher Scoring iterations: 8
```

b

```
# Function to estimate test error using the validation set approach
estimate_test_error <- function(data, seed) {
  set.seed(seed)

  # Split data into training and validation sets
  train_indices <- sample(seq_len(nrow(data)), size = 0.5 * nrow(data))
  train_set <- data[train_indices, ]
  validation_set <- data[-train_indices, ]

  # Fit logistic regression model on training data
  logistic_model <- glm(default ~ income + balance, data = train_set, family
= binomial)

  # Predict on validation data
  validation_probs <- predict(logistic_model, validation_set, type =
"response")
  validation_preds <- ifelse(validation_probs > 0.5, "Yes", "No")

  # Compute validation error
  validation_error <- mean(validation_preds != validation_set$default)

  return(validation_error)
}

# Estimate test error using different seeds
set.seed(1)  # Setting random seed
error1 <- estimate_test_error(Default, seed = 1)
error2 <- estimate_test_error(Default, seed = 2)
error3 <- estimate_test_error(Default, seed = 3)

# Print the validation errors
print(c(error1, error2, error3))

## [1] 0.0254 0.0238 0.0264
```

C

```
# Print the validation errors
validation_errors <- c(error1, error2, error3)
print(validation_errors)

## [1] 0.0254 0.0238 0.0264
```

D

```
# Function to estimate test error including the student variable
estimate_test_error_with_student <- function(data, seed) {
  set.seed(seed)
```

```r
  # Split data into training and validation sets
  train_indices <- sample(seq_len(nrow(data)), size = 0.5 * nrow(data))
  train_set <- data[train_indices, ]
  validation_set <- data[-train_indices, ]

  # Fit logistic regression model on training data including student variable
  logistic_model <- glm(default ~ income + balance + student, data =
train_set, family = binomial)

  # Predict on validation data
  validation_probs <- predict(logistic_model, validation_set, type =
"response")
  validation_preds <- ifelse(validation_probs > 0.5, "Yes", "No")

  # Compute validation error
  validation_error <- mean(validation_preds != validation_set$default)

  return(validation_error)
}

# Estimate test error using different seeds
error1_with_student <- estimate_test_error_with_student(Default, seed = 1)
error2_with_student <- estimate_test_error_with_student(Default, seed = 2)
error3_with_student <- estimate_test_error_with_student(Default, seed = 3)

# Print the validation errors including student variable
print(c(error1_with_student, error2_with_student, error3_with_student))

## [1] 0.0260 0.0246 0.0272
```

Problem 5

```r
# Install necessary packages if not already installed
if(!require(ISLR)) install.packages("ISLR", dependencies=TRUE)

# Load necessary libraries
library(ISLR)
library(MASS)

# Load the Default dataset
data("Default")
```

a

```r
# Fit the logistic regression model
logistic_model <- glm(default ~ income + balance, data = Default, family =
binomial)

# Summary of the model to get estimated coefficients and standard errors
summary(logistic_model)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

b

```r
# Define the boot.fn function
boot.fn <- function(data, index) {
  # Fit the logistic regression model on the subset of the data
  fit <- glm(default ~ income + balance, data = data, family = binomial,
subset = index)

  # Return the coefficients
  return(coef(fit))
}
```

C

```r
# Load the boot library
library(boot)

# Set a random seed for reproducibility
set.seed(1)

# Use the boot function to estimate standard errors
boot_results <- boot(data = Default, statistic = boot.fn, R = 1000)

# Print the bootstrap results
print(boot_results)

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
```

```
## boot(data = Default, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -3.945460e-02 4.344722e-01
## t2*  2.080898e-05  1.680317e-07 4.866284e-06
## t3*  5.647103e-03  1.855765e-05 2.298949e-04
```

d

```
# Extract the standard errors from the glm model
glm_se <- summary(logistic_model)$coefficients[, "Std. Error"]

# Extract the standard errors from the bootstrap results
boot_se <- apply(boot_results$t, 2, sd)

# Compare the results
results <- data.frame(
  Coefficient = names(glm_se),
  GLM_SE = glm_se,
  Bootstrap_SE = boot_se
)

print(results)

##              Coefficient       GLM_SE Bootstrap_SE
## (Intercept) (Intercept) 4.347564e-01 4.344722e-01
## income           income 4.985167e-06 4.866284e-06
## balance         balance 2.273731e-04 2.298949e-04
```

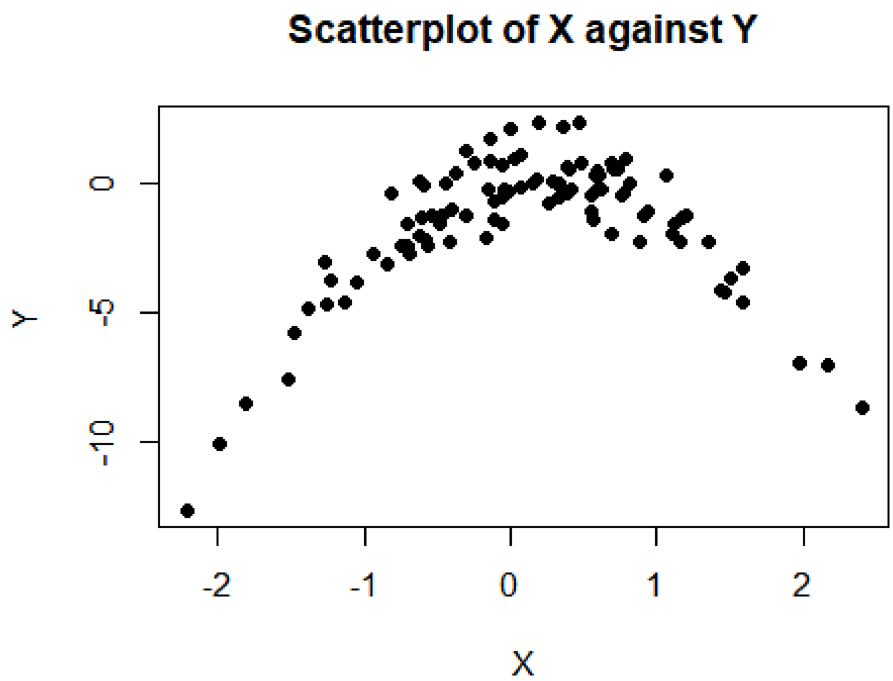
Problem 8

a

```
set.seed(1)
x <- rnorm(100)
y <- x - 2 * x^2 + rnorm(100)
```

b

```
# Create a scatterplot of X against Y
plot(x, y, main = "Scatterplot of X against Y", xlab = "X", ylab = "Y", pch =
19)
```



c

```
library(boot)

# Create the data frame
data <- data.frame(x = x, y = y)

# Define a function to compute the LOOCV error for a given model formula
compute_loocv_error <- function(formula, data) {
  model <- glm(formula, data = data)
  cv_result <- cv.glm(data, model, K = nrow(data))
  return(cv_result$delta[1])  # LOOCV error
}

# Set a random seed
set.seed(1)

# Compute LOOCV errors for the four models
loocv_error_1 <- compute_loocv_error(y ~ x, data)
loocv_error_2 <- compute_loocv_error(y ~ poly(x, 2), data)
loocv_error_3 <- compute_loocv_error(y ~ poly(x, 3), data)
loocv_error_4 <- compute_loocv_error(y ~ poly(x, 4), data)

# Print the LOOCV errors
loocv_errors <- c(loocv_error_1, loocv_error_2, loocv_error_3, loocv_error_4)
names(loocv_errors) <- c("Model 1", "Model 2", "Model 3", "Model 4")
print(loocv_errors)
```

```
##    Model 1    Model 2    Model 3    Model 4
## 7.2881616 0.9374236 0.9566218 0.9539049
```

d

```
# Set another random seed
set.seed(2)

# Compute LOOCV errors for the four models
loocv_error_1_seed2 <- compute_loocv_error(y ~ x, data)
loocv_error_2_seed2 <- compute_loocv_error(y ~ poly(x, 2), data)
loocv_error_3_seed2 <- compute_loocv_error(y ~ poly(x, 3), data)
loocv_error_4_seed2 <- compute_loocv_error(y ~ poly(x, 4), data)

# Print the LOOCV errors
loocv_errors_seed2 <- c(loocv_error_1_seed2, loocv_error_2_seed2,
loocv_error_3_seed2, loocv_error_4_seed2)
names(loocv_errors_seed2) <- c("Model 1", "Model 2", "Model 3", "Model 4")
print(loocv_errors_seed2)

##    Model 1    Model 2    Model 3    Model 4
## 7.2881616 0.9374236 0.9566218 0.9539049
```

e

```
# Determine the model with the smallest LOOCV error
best_model <- names(loocv_errors)[which.min(loocv_errors)]
print(best_model)

## [1] "Model 2"
```

f

```
# Fit each model and summarize
model_1 <- glm(y ~ x, data = data)
model_2 <- glm(y ~ poly(x, 2), data = data)
model_3 <- glm(y ~ poly(x, 3), data = data)
model_4 <- glm(y ~ poly(x, 4), data = data)

summary(model_1)

##
## Call:
## glm(formula = y ~ x, data = data)
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.6254     0.2619  -6.205 1.31e-08 ***
## x             0.6925     0.2909   2.380   0.0192 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.760719)
```

```
##
##     Null deviance: 700.85  on 99  degrees of freedom
## Residual deviance: 662.55  on 98  degrees of freedom
## AIC: 478.88
##
## Number of Fisher Scoring iterations: 2
```

```r
summary(model_2)
```

```
##
## Call:
## glm(formula = y ~ poly(x, 2), data = data)
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.5500     0.0958  -16.18  < 2e-16 ***
## poly(x, 2)1   6.1888     0.9580    6.46 4.18e-09 ***
## poly(x, 2)2 -23.9483     0.9580  -25.00  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9178258)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  89.029  on 97  degrees of freedom
## AIC: 280.17
##
## Number of Fisher Scoring iterations: 2
```

```r
summary(model_3)
```

```
##
## Call:
## glm(formula = y ~ poly(x, 3), data = data)
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09626 -16.102  < 2e-16 ***
## poly(x, 3)1   6.18883    0.96263   6.429 4.97e-09 ***
## poly(x, 3)2 -23.94830    0.96263 -24.878  < 2e-16 ***
## poly(x, 3)3   0.26411    0.96263   0.274    0.784
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9266599)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  88.959  on 96  degrees of freedom
## AIC: 282.09
##
## Number of Fisher Scoring iterations: 2
```

```r
summary(model_4)
```

```
##
## Call:
## glm(formula = y ~ poly(x, 4), data = data)
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, 4)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, 4)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, 4)3   0.26411    0.95905   0.275    0.784
## poly(x, 4)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

Problem 9

a

```r
# Load necessary libraries
library(ISLR2)
```

```
##
## Attaching package: 'ISLR2'

## The following objects are masked _by_ '.GlobalEnv':
##
##     Auto, Boston

## The following objects are masked from 'package:ISLR':
##
##     Auto, Credit

## The following object is masked from 'package:MASS':
##
##     Boston
```

```r
# Load the Boston dataset
data(Boston)

# Estimate the population mean of medv
mean_medv <- mean(Boston$medv)
mean_medv
```

```
## [1] 22.53281
```

b

```
# Estimate the standard error of the mean
n <- length(Boston$medv)
sample_sd <- sd(Boston$medv)
se_mean_medv <- sample_sd / sqrt(n)
se_mean_medv

## [1] 0.4088611
```

c

```
# Load the boot library
library(boot)

# Define a function for the bootstrap
bootstrap_mean <- function(data, indices) {
  return(mean(data[indices]))
}

# Perform the bootstrap
set.seed(1)  # For reproducibility
boot_results <- boot(Boston$medv, bootstrap_mean, R = 1000)

# Bootstrap estimate of standard error
boot_se_mean <- sd(boot_results$t)
boot_se_mean

## [1] 0.4106622
```

d

```
# 95% confidence interval using the bootstrap estimate
ci_bootstrap <- c(mean_medv - 2 * boot_se_mean, mean_medv + 2 * boot_se_mean)
ci_bootstrap

## [1] 21.71148 23.35413

# 95% confidence interval using t.test
ci_ttest <- t.test(Boston$medv)$conf.int
ci_ttest

## [1] 21.72953 23.33608
## attr(,"conf.level")
## [1] 0.95
```

e

```
# Estimate the population median of medv
median_medv <- median(Boston$medv)
median_medv

## [1] 21.2
```

f

```r
# Define a function to compute the median of medv
boot_median_fn <- function(data, index) {
  return(median(data[index]))
}

# Use the boot function to estimate the standard error
set.seed(1)
boot_median_results <- boot(data = Boston$medv, statistic = boot_median_fn, R
= 1000)
boot_median_results

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = boot_median_fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*     21.2 0.02295   0.3778075

# Extract the bootstrap estimate of the standard error for the median
boot_se_mu_med <- sd(boot_median_results$t)
boot_se_mu_med

## [1] 0.3778075
```

g

```r
# Estimate the tenth percentile of medv
mu_0.1 <- quantile(Boston$medv, 0.1)
mu_0.1

##   10%
## 12.75
```

h

```r
# Define a function to compute the tenth percentile of medv
boot_percentile_fn <- function(data, index) {
  return(quantile(data[index], 0.1))
}

# Use the boot function to estimate the standard error
set.seed(1)
boot_percentile_results <- boot(data = Boston$medv, statistic =
boot_percentile_fn, R = 1000)
boot_percentile_results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = boot_percentile_fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75  0.0339   0.4767526
```

```
# Extract the bootstrap estimate of the standard error for the tenth percentile
boot_se_mu_0.1 <- sd(boot_percentile_results$t)
boot_se_mu_0.1
```

```
## [1] 0.4767526
```