```
#p1
#(a)
library(ISLR)
data(Default)
names(Default)
summary(Default)
set.seed(1)
LG.fit = glm(default~income+balance, family=binomial)
summary(LG.fit)

#(b)
train_set = sample(dim(Default)[1],dim(Default)[1]/1.5)
#i. Split the sample set into a training set and a validation set.
NLG.fit = glm(default ~ income + balance, data = Default, family =
"binomial", subset = train_set)
#ii. Fit a logistic regression model using only the training data set.
probs = predict(NLG.fit, newdata = Default[-train_set, ], type =
"response")
pred.glm = rep("No", length(probs))
pred.glm[probs > 0.5] = "Yes"
#iii. Obtain a prediction of default status for each individual in the
validation set using a threshold of 0.5.
mean(pred.glm != Default[-train_set, ]$default)

#(c)
train_set = sample(dim(Default)[1],dim(Default)[1]/1.5)
NLG.fit = glm(default ~ income + balance, data = Default, family =
"binomial", subset = train_set)
probs = predict(NLG.fit, newdata = Default[-train_set, ], type =
"response")
pred.glm = rep("No", length(probs))
pred.glm[probs > 0.5] = "Yes"
mean(pred.glm != Default[-train_set, ]$default)

train_set = sample(dim(Default)[1],dim(Default)[1]/1.5)
NLG.fit = glm(default ~ income + balance, data = Default, family =
"binomial", subset = train_set)
probs = predict(NLG.fit, newdata = Default[-train_set, ], type =
"response")
pred.glm = rep("No", length(probs))
pred.glm[probs > 0.5] = "Yes"
mean(pred.glm != Default[-train_set, ]$default)

train_set = sample(dim(Default)[1],dim(Default)[1]/1.5)
NLG.fit = glm(default ~ income + balance, data = Default, family =
"binomial", subset = train_set)
probs = predict(NLG.fit, newdata = Default[-train_set, ], type =
"response")
pred.glm = rep("No", length(probs))
pred.glm[probs > 0.5] = "Yes"
mean(pred.glm != Default[-train_set, ]$default)

#(d)adding one student variable, check the test error
train_set <- sample(dim(Default)[1], dim(Default)[1] / 1.5)
LG.glm <- glm(default ~ income + balance + student, data = Default,
family = "binomial", subset = train_set)
pred.glm <- rep("No", length(probs))
```

```
probs <- predict(LG.glm, newdata = Default[-train_set, ], type =
"response")
pred.glm[probs > 0.5] <- "Yes"
mean(pred.glm != Default[-train_set, ]$default)


#p2
#(a) Generate a simulated data set as follows:
x=rnorm(200)
y=x-2*x^2+rnorm(200)
#(b)
plot(x,y)
#(c)
library(boot)
Data = data.frame(x,y)
set.seed(1)
LG.fit = glm(y~x)
cv.glm(Data, LG.fit)$delta

LG2.fit = glm(y~poly(x,2))
cv.glm(Data, LG2.fit)$delta

LG3.fit = glm(y~poly(x,3))
cv.glm(Data, LG3.fit)$delta

LG4.fit = glm(y~poly(x,4))
cv.glm(Data, LG4.fit)$delta

#(d)
set.seed(3)
LG.fit = glm(y~x)
cv.glm(Data, LG.fit)$delta

LG.fit = glm(y~poly(x,2))
cv.glm(Data, LG.fit)$delta

LG.fit = glm(y~poly(x,3))
cv.glm(Data, LG.fit)$delta

LG.fit = glm(y~poly(x,4))
cv.glm(Data, LG.fit)$delta

#(f)
# Generate simulated data set
set.seed(1)
x <- rnorm(200)
y <- x - 2*x^2 + rnorm(200)

# Define the four models
model1 <- lm(y ~ x)
model2 <- lm(y ~ x + I(x^2))
model3 <- lm(y ~ x + I(x^2) + I(x^3))
model4 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4))

# Create a function to perform 5-fold cross-validation and return the
cross-validation error
cv <- function(model) {
  cv.error <- rep(NA, 5)
```

```
  folds <- cut(seq_along(x), breaks = 5, labels = FALSE)
  for (i in 1:5) {
    test.index <- which(folds == i)
    train.index <- which(folds != i)
    cv.error[i] <- mean((y[test.index] - predict(model, newdata =
data.frame(x = x[test.index])))^2)
  }
  mean(cv.error)
}

# Compute the cross-validation errors for the four models
cv.error1 <- cv(model1)
cv.error2 <- cv(model2)
cv.error3 <- cv(model3)
cv.error4 <- cv(model4)

# Print the cross-validation errors
cat("CV error for model 1:", cv.error1, "\n")
cat("CV error for model 2:", cv.error2, "\n")
cat("CV error for model 3:", cv.error3, "\n")
cat("CV error for model 4:", cv.error4, "\n")

#(g)
set.seed(3)
x <- rnorm(200)
y <- x - 2*x^2 + rnorm(200)

# Define the four models
model1_10 <- lm(y ~ x)
model2_10 <- lm(y ~ x + I(x^2))
model3_10 <- lm(y ~ x + I(x^2) + I(x^3))
model4_10 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4))

# Create a function to perform 10-fold cross-validation and return the
cross-validation error
cv <- function(model) {
  cv.error <- rep(NA, 10)
  folds <- cut(seq_along(x), breaks = 10, labels = FALSE)
  for (i in 1:10) {
    test.index <- which(folds == i)
    train.index <- which(folds != i)
    cv.error[i] <- mean((y[test.index] - predict(model, newdata =
data.frame(x = x[test.index])))^2)
  }
  mean(cv.error)
}

# Compute the cross-validation errors for the four models
cv.error1_10 <- cv(model1_10)
cv.error2_10 <- cv(model2_10)
cv.error3_10 <- cv(model3_10)
cv.error4_10 <- cv(model4_10)

# Print the cross-validation errors
cat("CV error for model 1:", cv.error1_10, "\n")
cat("CV error for model 2:", cv.error2_10, "\n")
cat("CV error for model 3:", cv.error3_10, "\n")
cat("CV error for model 4:", cv.error4_10, "\n")
```