A1

Q1

```r
h1q1 <- read.csv("h1q1.csv")
set.seed(42)

rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}

get_complexity = function(model) {
  length(coef(model)) - 1
}

make_answer = function(model) {
  train_rmse = get_rmse(model, data=train_data, response="y")
  cat("train rmse", train_rmse, "\n")

  test_rmse = get_rmse(model, data=test_data, response="y")
  cat("test rmse", test_rmse, "\n")

  complexity = get_complexity(model)
  cat("complexity", complexity, "\n")
}

train_index = sample(1:nrow(h1q1), size=round(0.5*nrow(h1q1)))
train_data = h1q1[train_index,]
test_data = h1q1[-train_index,]

# Just a linear model
m1 <- y ~ .
m2 <- y ~ . + I(a^2) + I(b^2) + I(c^2)
m3 <- y ~ .^2 + I(a^2) + I(b^2) + I(c^2)
m4 <- y ~ a * b * c * d + I(a^2) + I(b^2) + I(c^2)

fit1 = lm(m1, data=train_data)
make_answer(fit1)

fit2 = lm(m2, data=train_data)
make_answer(fit2)

fit3 = lm(m3, data=train_data)
make_answer(fit3)

fit4 = lm(m4, data=train_data)
make_answer(fit4)
```

Q2

```r
# install.packages(c("readr", "tibble"))

library(readr)
library(tibble)
library(MASS)
data(Boston)
Boston = as_tibble(Boston)

rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}

get_rmse = function(model, data, response) {
  rmse(actual = subset(data, select = response, drop = TRUE),
       predicted = predict(model, data))
}

get_complexity = function(model) {
  length(coef(model)) - 1
}

set.seed(42)
boston_index = sample(1:nrow(Boston), size = 400)
train_boston = Boston[boston_index, ]
test_boston = Boston[-boston_index, ]

fit = lm(medv ~ .^2, data = train_boston)
fit_smaller = lm(medv ~ ., data = train_boston)
fit_larger = lm(medv ~ .^2 + I(black^2), data=train_boston)

get_rmse(fit, data=train_boston, response="medv")
get_rmse(fit_smaller, data=train_boston, response="medv")
get_rmse(fit_larger, data=train_boston, response="medv")

get_rmse(fit, data=test_boston, response="medv")
get_rmse(fit_smaller, data=test_boston, response="medv")
get_rmse(fit_larger, data=test_boston, response="medv")

get_complexity(fit)
get_complexity(fit_smaller)
get_complexity(fit_larger)
```

Q3

```r
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}

h1q3_test <- read.csv("h1q3-test-data.csv")
```

```r
h1q3_train <- read.csv("h1q3-train-data.csv")

for (k in seq(5, 50, by=5)) {
  cat("k:", k, "\t")
  pred_train = FNN::knn.reg(train = h1q3_train["x"],
    test = h1q3_train["x"], y = h1q3_train$y, k = k)$pred
  train_rmse = rmse(predicted = pred_train, actual = h1q3_train$y)

  pred_test = FNN::knn.reg(train = h1q3_train["x"],
    test = h1q3_test["x"], y = h1q3_train$y, k = k)$pred
  test_rmse = rmse(predicted = pred_test, actual = h1q3_test$y)

  cat("train rmse:", train_rmse, "\ttest rmse:", test_rmse, "\n")
}
```

Q4

```r
mse = function(actual, predicted) {
  mean((actual - predicted) ^ 2)
}

h1q4_test <- read.csv("h1q4-test-data.csv")
h1q4_train <- read.csv("h1q4-train-data.csv")

# do for k = 1, 5, 25
for (k in list(1, 5, 25)) {
  predTest = FNN::knn.reg(train = h1q4_train[, -1],
    test = h1q4_test[, -1], y = h1q4_train$y, k = k)$pred
  test_mse = mse(predicted = predTest, actual = h1q4_test$y)

  predScaled = FNN::knn.reg(train = scale(h1q4_train[, -1]),
    test = scale(h1q4_test[, -1]), y = h1q4_train$y, k = k)$pred
  scaled_test_mse = mse(predicted = predScaled, actual = h1q4_test$y)

  cat("k:", k)
  cat("\ttest mse:", test_mse, "\tscaled test mse:", scaled_test_mse, "\n")
}
```

Q5

```r
# install.packages("ISLR")
library(ISLR)

rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}

get_rmse = function(model, data, response) {
  rmse(actual = subset(data, select = response, drop = TRUE),
       predicted = predict(model, data))
}

auto = Auto[, !names(Auto) %in% c("name")]
set.seed(42)
auto_idx = sample(1:nrow(auto), size = round(0.5 * nrow(auto)))
auto_trn = auto[auto_idx, ]
auto_tst = auto[-auto_idx, ]

model = lm(mpg ~ ., data = auto_trn)

get_rmse(model, data=test_data, response="mpg")

train_index = sample(1:nrow(auto), size=round(0.5*nrow(auto)))
train_data = auto[train_index,]
test_data = auto[-train_index,]

pred = FNN::knn.reg(train = scale(train_data[, -1]),
  test = scale(test_data[, -1]), y = train_data$y, k = 6)$pred
rmse(predicted = pred, actual = test_data$y)
```