Stat 154 HW 05

Mokssh Surve

3/3/2020

```
set.seed(12345)
```

Problem 3

a)

```
full_data <- read.csv("nba-teams-2019.csv")
keeps_1 <- c('W', 'PTS')
keeps_2 <- c('W', 'PTS', 'FGM', 'X3PM', 'FTM', 'AST')
keeps_3 <- c('W', 'PTS', 'FGM', 'X3PM', 'FTM', 'AST', 'OREB', 'DREB', 'TOV', 'STL', 'BLK')


# Model 1
dat_1 <- full_data[keeps_1]
model_1 <- lm(W ~ PTS, data=dat_1)
mse_1 <- mean(model_1$residuals^2)
mse_1
```

```
## [1] 78.80752
```

```
# Model 2
dat_2 <- full_data[keeps_2]
model_2 <- lm(W ~. , data=dat_2)
mse_2 <- mean(model_2$residuals^2)
mse_2
```

```
## [1] 65.67837
```

```
# Model 3
dat_3 <- full_data[keeps_3]
model_3 <- lm(W ~., data=dat_3)
mse_3 <- mean(model_3$residuals^2)
mse_3
```

```
## [1] 35.01552
```

Model 3 has the best (lowest) MSE. This is expected owing to the greater number of predictor variables associated with this model. In addition, it should be noted that since this is the *"in sample"* MSE, a greater number of predictors in this model allows there to be a better fit to the data set the regression is carried on.

b) & c)

```r
#length of each fold
len_fold <- nrow(full_data)/10

# MODEL 1
perm_1 <- dat_1[sample(nrow(dat_1)), ]
fold_mse_1 <- c()
for (i in 1:10) {

  low <- (3*i) - 2
  high <- ((3*i))
  temp_model <- lm(W ~ PTS, data=perm_1[-c(low:high), ])
  predictions <- predict(temp_model, data=perm_1[c(low:high), ]$PTS)
  temp_mse <- mean((predictions - perm_1[c(low:high), ]$W)^2)
  fold_mse_1 <- c(fold_mse_1, temp_mse)
}
cv_mse_1 <- mean(fold_mse_1)


# MODEL 2
perm_2 <- dat_2[sample(nrow(dat_2)), ]
fold_mse_2 <- c()
for (i in 1:10) {

  low <- (3*i) - 2
  high <- ((3*i))
  temp_model <- lm(W ~. , data=perm_2[-c(low:high), ])
  predictions <- predict(temp_model, data=perm_2[c(low:high), -c(1)])
  temp_mse <- mean((predictions - perm_2[c(low:high), ]$W)^2)
  fold_mse_2 <- c(fold_mse_2, temp_mse)
}
cv_mse_2 <- mean(fold_mse_2)


# MODEL 3
perm_3 <- dat_3[sample(nrow(dat_3)), ]
fold_mse_3 <- c()
for (i in 1:10) {

  low <- (3*i) - 2
  high <- ((3*i))
  temp_model <- lm(W ~. , data=perm_3[-c(low:high), ])
  predictions <- predict(temp_model, data=perm_3[c(low:high), -c(1)])
  temp_mse <- mean((predictions - perm_3[c(low:high), ]$W)^2)
  fold_mse_3 <- c(fold_mse_3, temp_mse)
}
cv_mse_3 <- mean(fold_mse_3)


cv_mse_1
```

```
## [1] 220.3988
```

```
cv_mse_2
```

```
## [1] 235.8132
```

```
cv_mse_3
```

```
## [1] 259.6193
```

d)

Model 1 has the best CV MSE even though Model 3 had the best in-sample MSE. This is possible and highly likely in this case in pareticular because the 10-fold mechanism leaves only 3 data points in the test-set => that there is a lot of scope for variation. In addition, the random reshuffling of the rows means that this value can fluctuate quite drastically in this particular case.

Problem 4

It is important to note that the larger the split in favour of the training set, the lesser the error associated with the training error will be - owing to the overfitting of the data relative to the test set. This would subsequently mean that for instance, a (80,20) split in favor of the training set would have a larger test error, & smaller training error than a (20,80) split.

a)

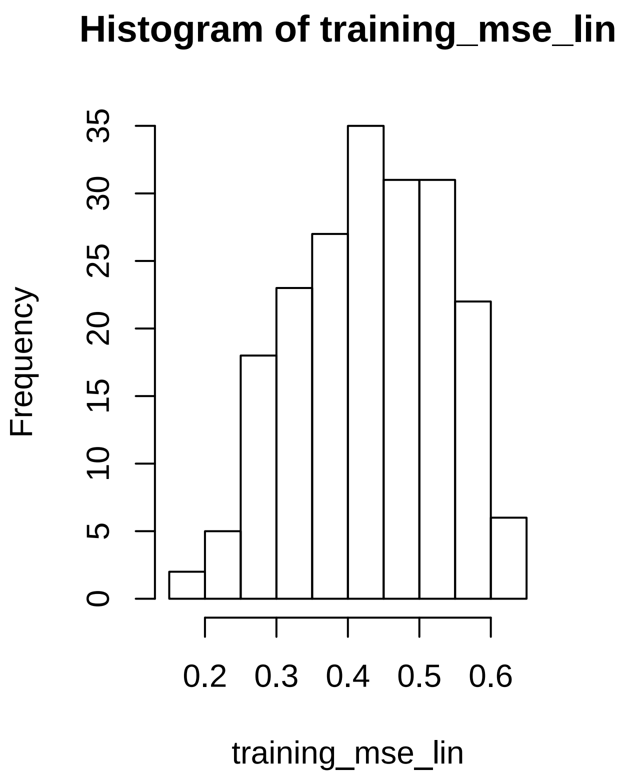
I would expect the linear regression to fit better since even though x~U(0,1), y is linear in x => the training and test error is expected to be better (lesser) for the linear regression as opposed to the quadratic regression.

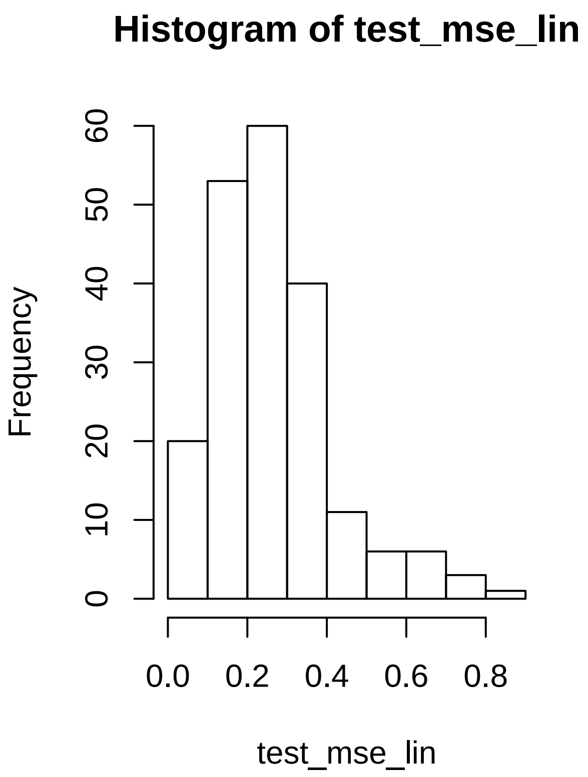
b)

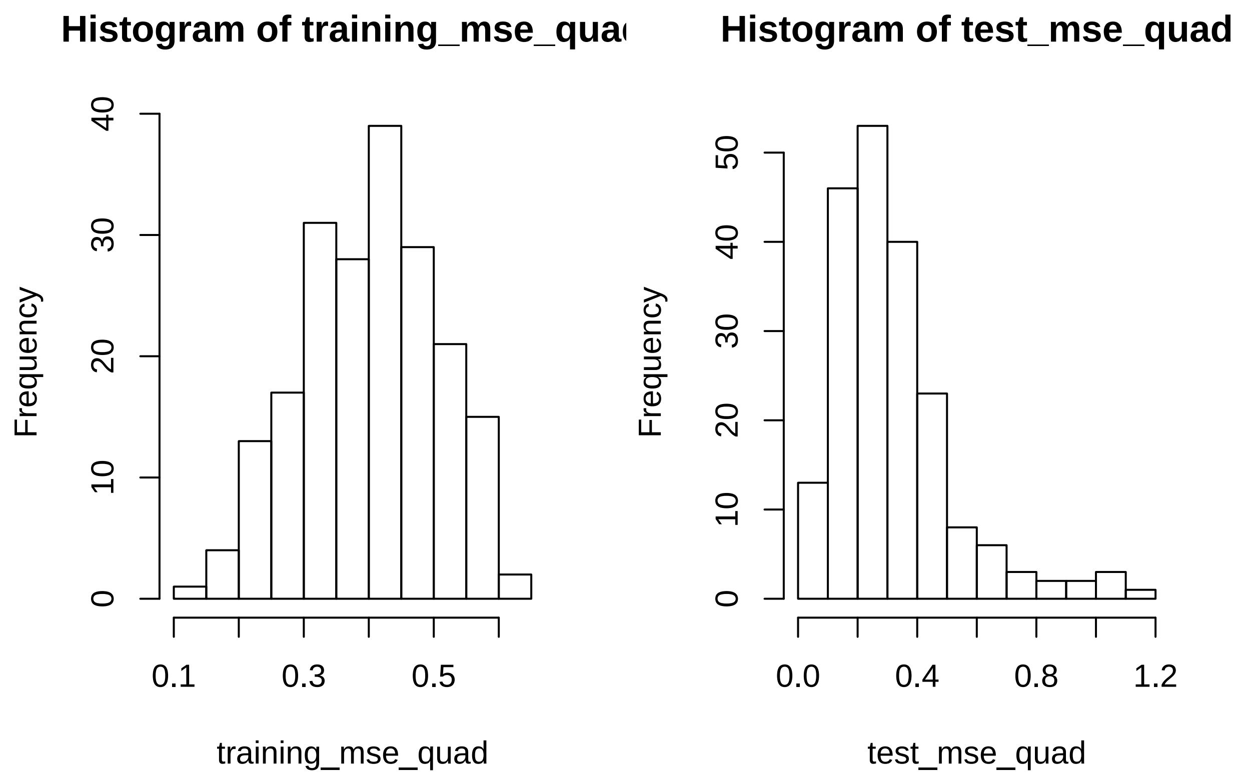
As mentioned before, it is expected that as the split increases in favour of the training set (i.e from a (30,70) to (70,30)), the training error will decreason and the test error will likely increase, and vice versa. This is likely to be the case since a greater split for the training set will most likely imply a better/possibly overfit model => that the training error will be lower. The test error on the other hand is likely to become higher since the smaller set leaves room for more randomness (since the split is randomised).

c) & d)

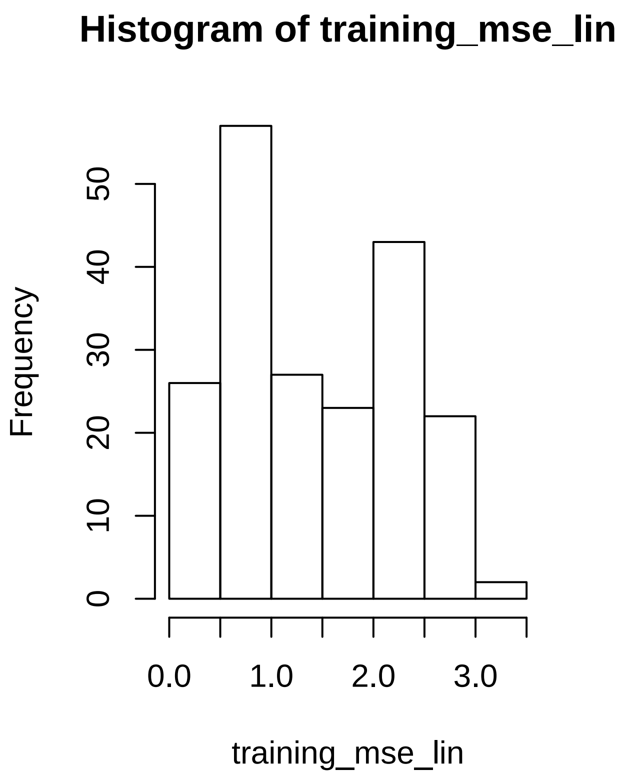
```
mse_calc <- function(training_split){

      x <- runif(20, 0, 1)
      e <- rnorm(20, mean=0, sd=1)
      len <- length(x)

      training_mse_lin <- rep(0, 200)
      test_mse_lin <- rep(0, 200)
      training_mse_quad <- rep(0, 200)
      test_mse_quad <- rep(0, 200)
      for (i in 1:200){

        train <- training_split/100
        training_x <- sample(x, floor(len * train))
        training_e <- sample(e, floor(len * train))
        training_y <- training_x + training_e
        test_x <- x[!(x %in% training_x)]
        test_e <- x[!(e %in% training_e)]
        test_y <- test_x + test_e

        # linear model fit
        model_linear <- lm(training_y ~ training_x)
        summ_linear_training <-  summary(model_linear)
        training_mse_lin[i] <- mean(summ_linear_training$residuals^2)
        test_mse_lin[i] <- mean( (test_y - predict.lm(model_linear, data.frame(training_x=test_x)))^2 )

        # quadratic model fit
        model_quadratic <- lm(training_y ~ poly(training_x, 2))
        summ_quadratic_training <- summary(model_quadratic)
        training_mse_quad[i] <- mean(summ_quadratic_training$residuals^2)
        test_mse_quad[i] <- mean( (test_y - predict.lm(model_quadratic, data.frame(training_x=test_x)))


      }

      # Linear Model
      par(mfrow=c(1,2))
      hist(training_mse_lin)
      hist(test_mse_lin)

      # Quadratic Model
      par(mfrow=c(1,2))
      hist(training_mse_quad)
      hist(test_mse_quad)

}

mse_calc(70)
```

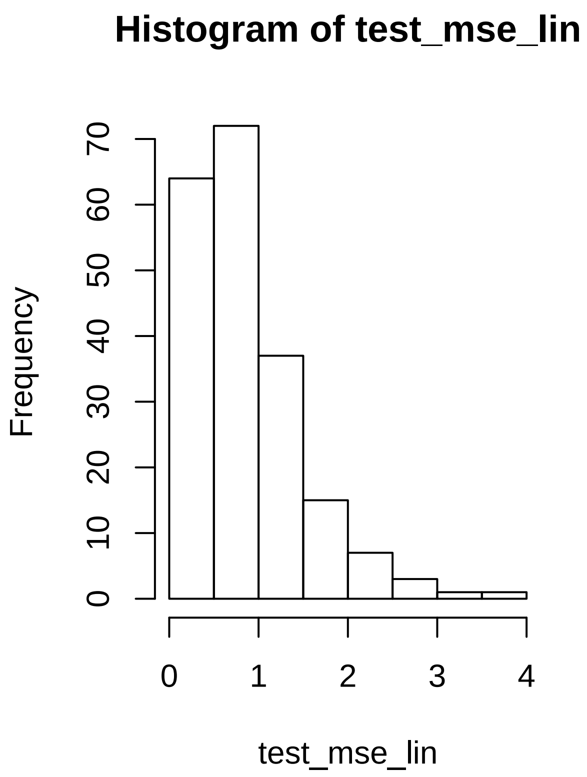

Histogram of training_mse_lin
Frequency
0
5
10
15
20
25
30
35
0.2
0.3
0.4
0.5
0.6
training_mse_lin

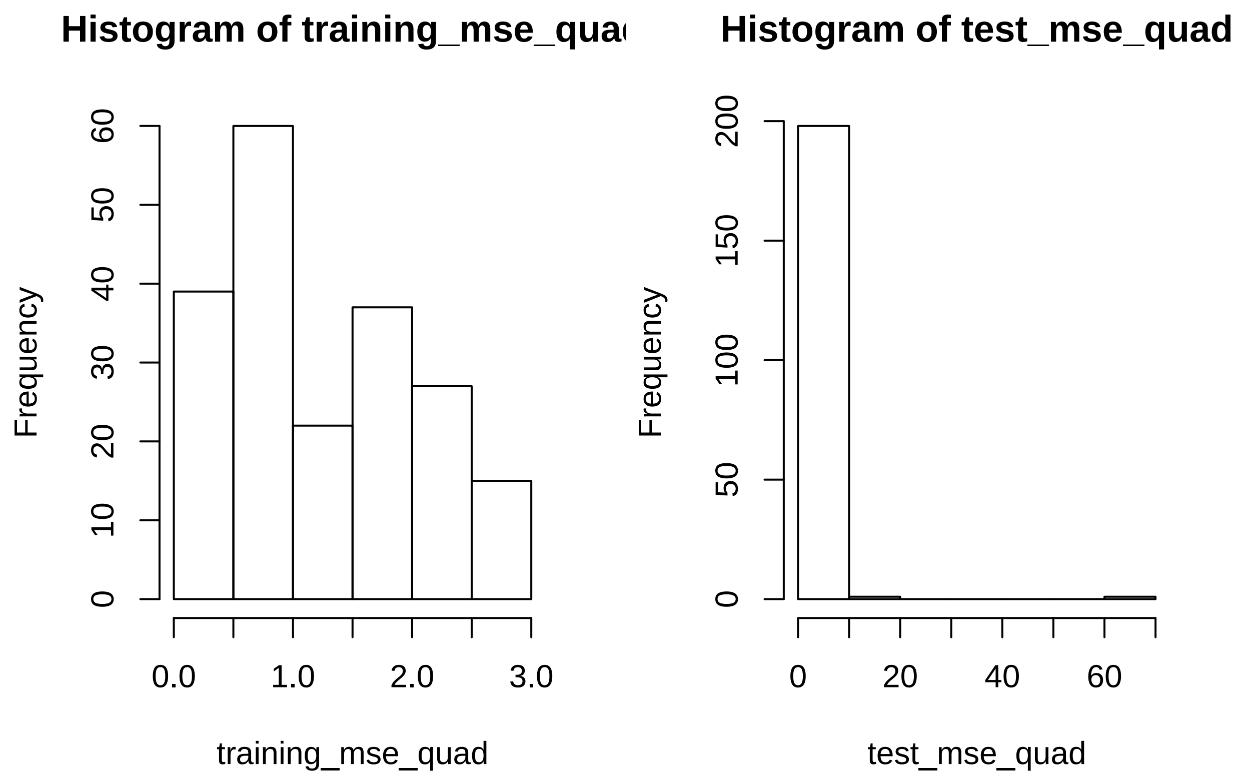

Histogram of test_mse_lin
Frequency
0
10
20
30
40
50
60
0.0
0.2
0.4
0.6
0.8
test_mse_lin


Histogram of training_mse_qua
Frequency
40
30
20
10
0
0.1
0.3
0.5
training_mse_quad
Histogram of test_mse_quad
Frequency
50
40
30
20
10
0
0.0
0.4
0.8
1.2
test_mse_quad

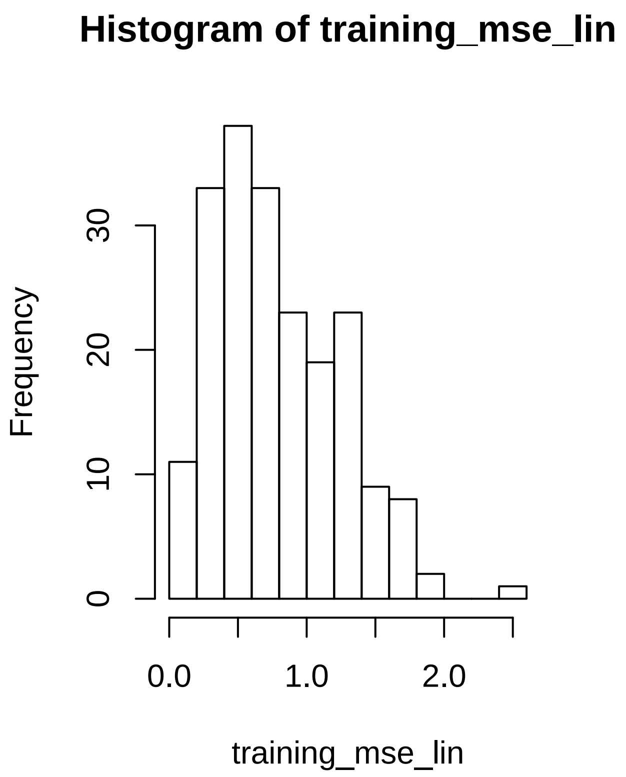
```
mse_calc(50)
```

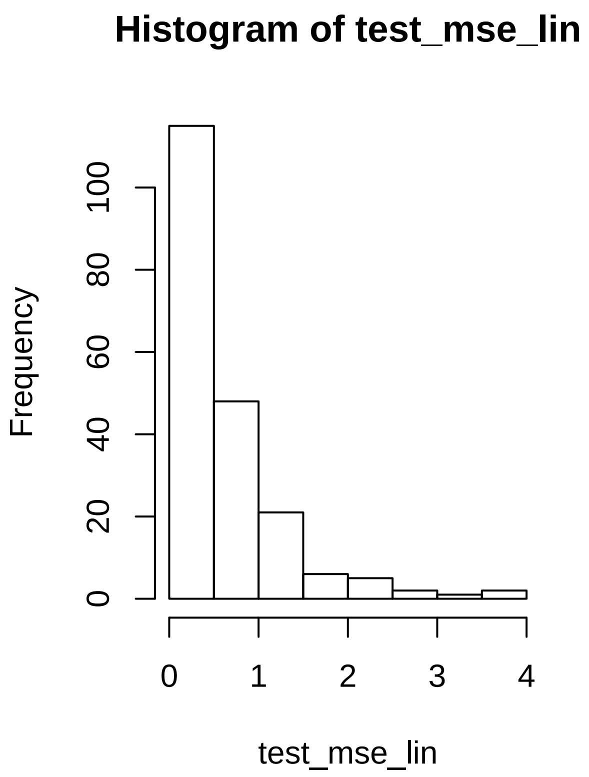

Histogram of training_mse_lin
Frequency
training_mse_lin
0
10
20
30
40
50
0.0
1.0
2.0
3.0

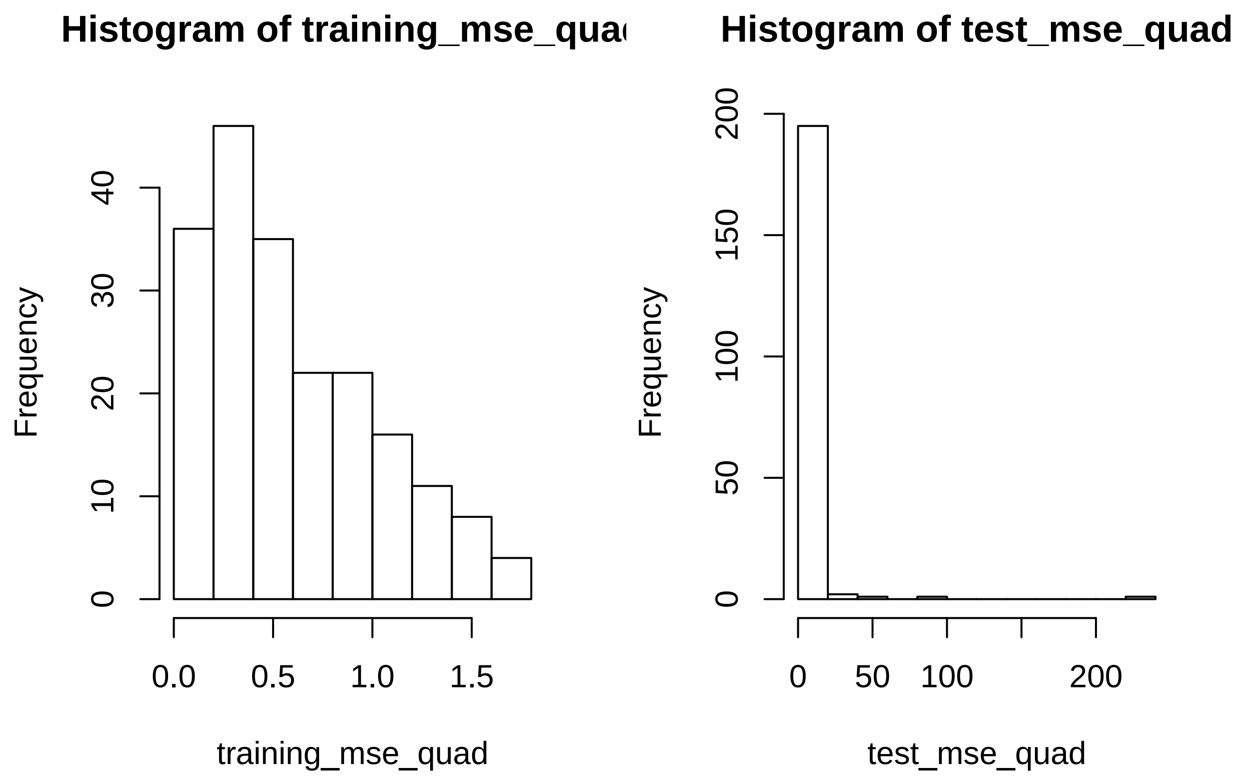

Histogram of test_mse_lin
Frequency
test_mse_lin
0
10
20
30
40
50
60
70
0
1
2
3
4


Histogram of training_mse_qua
Frequency
training_mse_quad
Histogram of test_mse_quad
Frequency
test_mse_quad

```r
mse_calc(30)
```


Histogram of training_mse_lin
Frequency
0
10
20
30
0.0
1.0
2.0
training_mse_lin


Histogram of test_mse_lin
Frequency
0
20
40
60
80
100
0
1
2
3
4
test_mse_lin



e)

As was predicted, as the split decreases in favour of the training set, the training MSE increases by a considerable amount, and the test MSE decreases by a small amount. This is seen clearly in the histogram plots.

Problem 5

a)

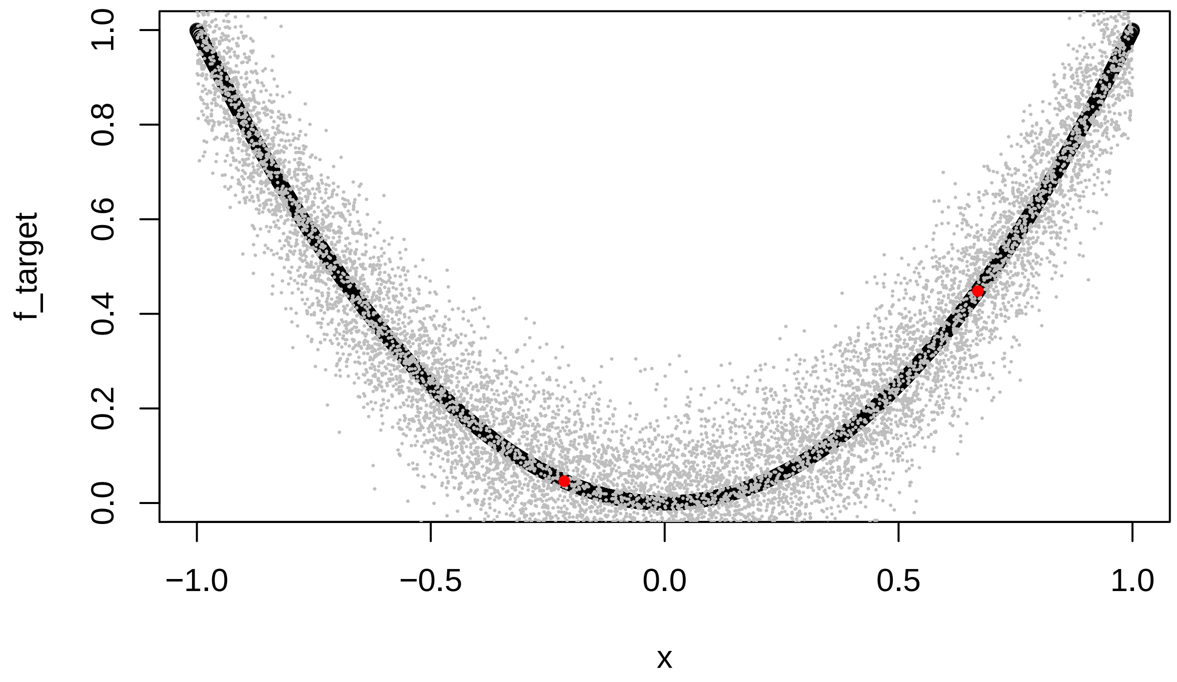
Based on the information provided in the write-up, the following experiment was devised by me in order to numerically find determine the average hypothesis fit for each of the hypotheses, and the overall estimated MSE for them as well:
We are given:

- **x~U(-1,1)**
- **error~N(0, sigma^2)** Note: sigma (sd) is taken to be 0.1
- **y = x^2 + error**

Based on this information, 10,000 uniformly distributed (for x) and 10,000 normally distributed (for error) numbers are randomly generated to serve as the complete test set. In addition, for a set 200 simulations, 2 different training data set points are generated along the same thought process - keeping in mind whether

the scenario being tested on is noisy (error=/0) or noiseless (error=0) scenarios.
For each of these simulations (200), the fit is plotted (green lines) and along all of these, the averages are determined for each of the hypotheses in order to get an **average hypothesis (red line)** (averaged out over the 200 simulations).
Subsequently, the **overall expected MSE** can be computed for each of the noisy/noiseless scenarios for both H0 and H1. => a total of 4 overall expected MSEs as averaged out over all the 10,000 randomly generated points (along the distributions as specified).

```r
set.seed(12345)
x <- runif(10000, -1, 1)
e <- rnorm(10000, mean=0, sd=0.1)
f_target <- x^2
f_target_noisy <- f_target + e

#training dataset <- Noiseless
x_d <- runif(2, -1, 1)
y_d <- x_d^2

#training dataset <- Noisy
e_d <- rnorm(2, mean=0, sd=0.1)
y_d_noisy <- y_d + e_d

plot(x, f_target, lwd=0.01)
points(x, f_target_noisy, cex=0.1, col='grey')
points(x_d, y_d, cex=0.8, col='red', pch=16)
```



Functions that I'll use

```r
line_equation <- function (p1, p2) {
  slope <- (p2[2]-p1[2]) / (p2[1]-p1[1])
  intercept <- p1[2] - slope*(p1[1])
  list(slope=slope, intercept=intercept)
}

# if parameter noisy=1, then noise included
# if paramter noisy=0, then no noise present
fit <- function(hypothesis_number, noisy) {

  # getting a new training set D
  x_d <- runif(2, -1, 1)
  e <- rnorm(2, mean=0, sd=0.1)

  if (noisy==0)
  {y_d <- (x_d)^2}
  else
  {y_d <- (x_d)^2 + e}

  # developing & plotting the hypothesis based on fit
  if (hypothesis_number==0)
    {h_fit <- sum(y_d)/2}
  else
    {h_fit <- line_equation(c(x_d[1], y_d[1]), c(x_d[2], y_d[2]))}

  return (h_fit)
}
```

b)

Noiseless scenario

```r
sims <- 200
temp <- 0
hyp_0_vec <- c()
# Hypothesis 0
for (i in 1:sims) {
  hyp_0 <- fit(0, 0)
  hyp_0_vec <- c(hyp_0_vec, hyp_0)
}
g_bar_0 <- mean(hyp_0_vec)
bias_sq_0 <- round(mean((g_bar_0 - f_target)^2), 4)
var_0 <- round(mean((hyp_0_vec - g_bar_0)^2), 4)
sig <- 0 #since no error (noise) term for noiseless scenario
overall_mse_0_noiseless <- bias_sq_0 + var_0 + sig
overall_mse_0_noiseless
```

```
## [1] 0.1279
```

```r
#Hypothesis 1
b0_hat <- rep(0, 200)
b1_hat <- rep(0, 200)
for (i in 1:sims) {
  hyp_1 <- fit(1,0)
  b0_hat[i] <- unlist(hyp_1[1])
  b1_hat[i] <- unlist(hyp_1[2])
}
b0_bar <- mean(b0_hat)
b1_bar <- mean(b1_hat)
g_bar_1 <- list(intercept = b0_bar, coefficient = b1_bar)


bias_sq_1 <- round(mean((b0_bar + b1_bar*x - f_target)^2), 4)
var_1_x <- (b0_hat + b1_hat*x)^2 - 2*(b0_hat + b1_hat*x) * (b0_bar + b1_bar * x) + (b0_bar + b1_bar*x)^
var_1 <- round(mean(var_1_x), 4)
sig <- 0 #since no error (noise) term for noiseless scenario
overall_mse_1_noiseless <- bias_sq_1 + var_1 + sig
overall_mse_1_noiseless
```

```
## [1] 0.9168
```

Noisy scenario

```r
sims <- 200
temp <- 0

hyp_0_vec <- c()
# Hypothesis 0
for (i in 1:sims) {
  hyp_0 <- fit(0, 1)
  hyp_0_vec <- c(hyp_0_vec, hyp_0)
}
g_bar_0 <- mean(hyp_0_vec)
bias_sq_0 <- round(mean((g_bar_0 - f_target)^2), 4)
var_0 <- round(mean((hyp_0_vec - g_bar_0)^2), 4)
sig <- 0.1^2 #since sigma was assumed to be 0.1 => the variance of the noise is sigma^2 = 0.1^2
overall_mse_0_noisy <- bias_sq_0 + var_0 + sig
overall_mse_0_noisy
```

```
## [1] 0.1423
```

```r
#Hypothesis 1
b0_hat <- rep(0, 200)
b1_hat <- rep(0, 200)
for (i in 1:sims) {
  hyp_1 <- fit(1,1)
  b0_hat[i] <- unlist(hyp_1[1])
  b1_hat[i] <- unlist(hyp_1[2])
}
```

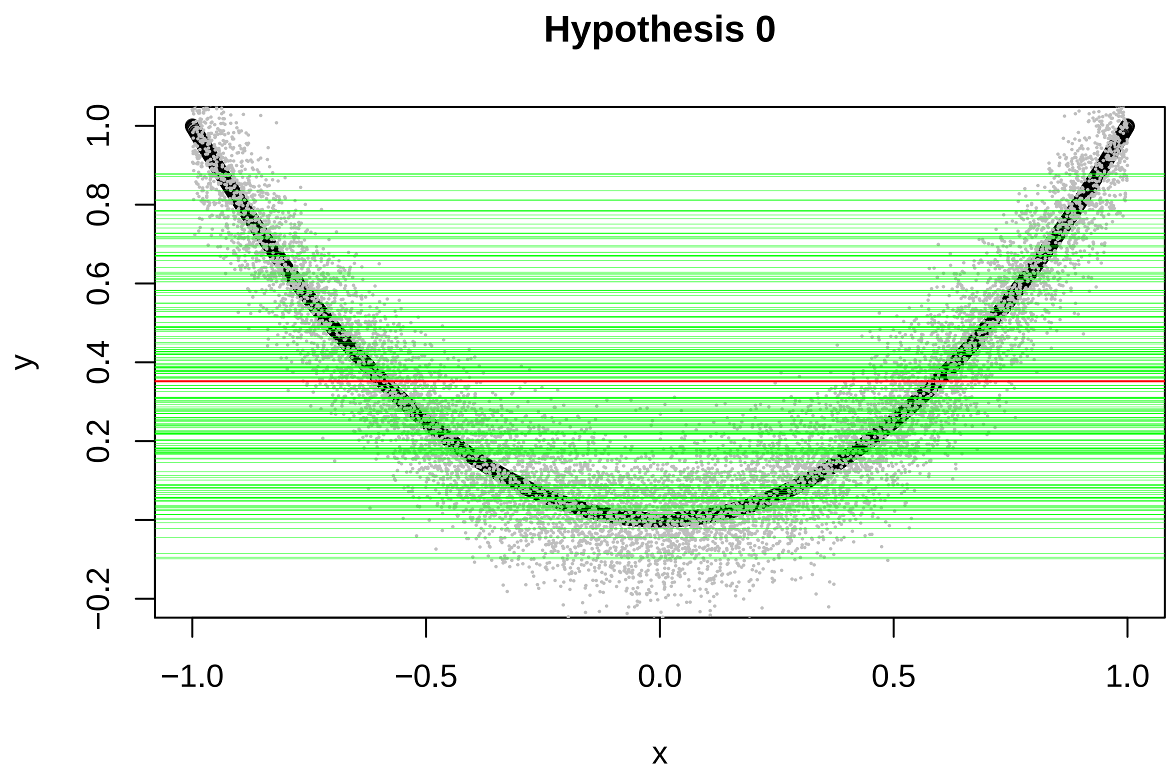
```r
b0_bar <- mean(b0_hat)
b1_bar <- mean(b1_hat)
g_bar_1 <- list(intercept = b0_bar, coefficient = b1_bar)

bias_sq_1 <- round(mean((b0_bar + b1_bar*x - f_target)^2), 4)
var_1_x <- (b0_hat + b1_hat*x)^2 - 2*(b0_hat + b1_hat*x) * (b0_bar + b1_bar * x) + (b0_bar + b1_bar*x)^
var_1 <- round(mean(var_1_x), 4)
sig <- 0.1^2 #since sigma was assumed to be 0.1 => the variance of the noise is sigma^2 = 0.1^2
overall_mse_1_noisy <- bias_sq_1 + var_1 + sig
overall_mse_1_noisy
```

```
## [1] 2.2976
```

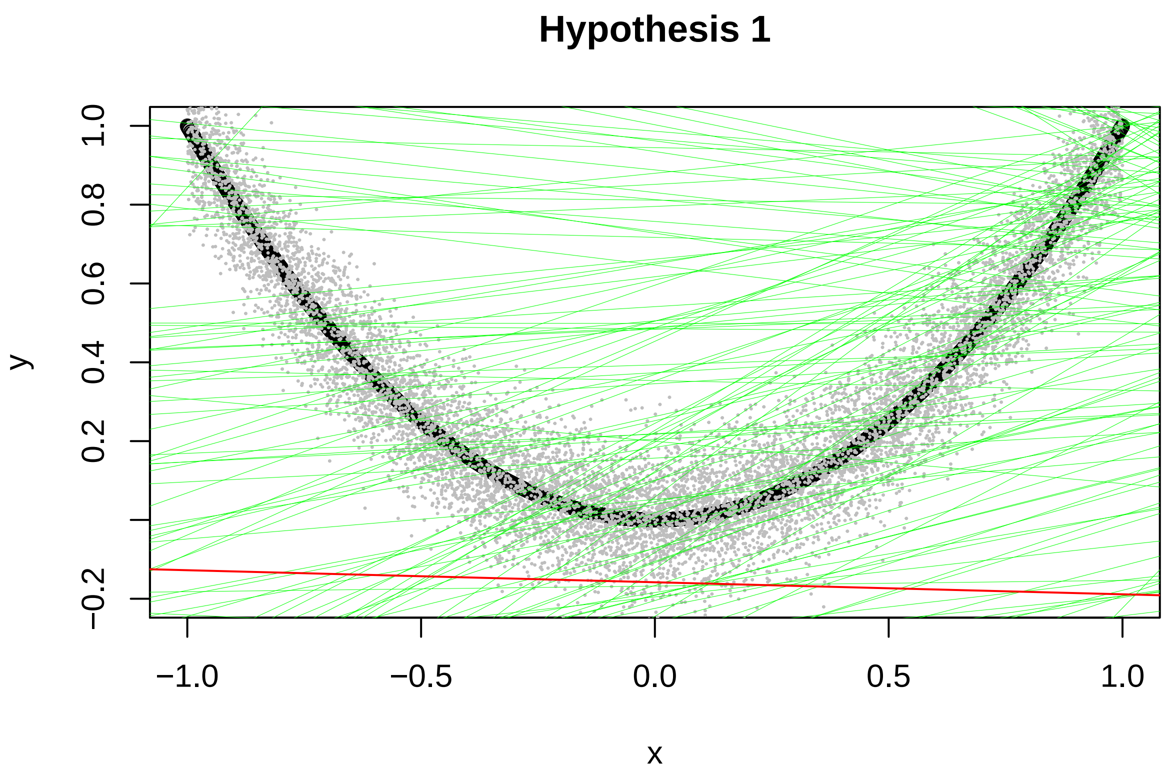
c) Plots (Noisy Scenario)

The Function & Function Call

```r
# Plot for Hypothesis 0
hyp_0 <- 0
total_hyp_0 <- 0
plot(x, f_target, lwd=0.01, main = 'Hypothesis 0', ylab = 'y', ylim = c(-0.2,1))
points(x, f_target_noisy, cex=0.1, col='grey')
for (i in 1:200){
  hyp_0 <- fit(0, 1)
  abline(h=hyp_0, col='green', lwd=0.1)
  total_hyp_0 <- total_hyp_0 + hyp_0
}
abline(h = total_hyp_0/200, col='red')
```



```
# Plot for Hypothesis 1
b0_hat <- rep(0, 200)
b1_hat <- rep(0, 200)
plot(x, f_target, lwd=0.01, main = 'Hypothesis 1', ylab = 'y', ylim = c(-0.2,1))
points(x, f_target_noisy, cex=0.1, col='grey')
for (i in 1:200){
  hyp_1 <- fit(1, 1)
  b0_hat[i] <- unlist(hyp_1[1])
  b1_hat[i] <- unlist(hyp_1[2])
  abline(a = b0_hat[i], b = b1_hat[i], col='green', lwd=0.1)
}
b0_bar <- mean(b0_hat)
b1_bar <- mean(b1_hat)
abline(a=b0_bar, b=b1_bar, col='red', lwd=1)
```



d)

The overall expected MSE was found to be **greater for the H1 than it was for H0** - for both the noiseless and noisy scenarios. As can be seen in each of the plots, the following legend can be deduced:

- **Solid Black Line:** Hypothetical signal (noiseless)
- **Grey Points:** Noisy y-values
- **Green Lines:** Multiple fits based on the varying training data set
- **Red Line:** Average Fit for respective hypothesis (based on plot)

As is evident from the plots, the H0 fit is more centrally placed and provides a "better" estimate of the data points - thus culminating into a smaller overall estimated MSE for both the noisy and noiseless scenarios. In addition, there seems to be a relative overfitting of the data in the case of H1 based on the training data set (D) the green lines correspond to.