MAST09104: Introduction to Statistical Learning

Week 4 Workshop and Lab Solutions

Workshop questions

Note that some of these involve R.

1. David asked a question about the green plane picture in Figure 1 in Module 4: namely, whether the data vector y could be orthogonal to the plane of the column space of X,.

   To answer this, let $H = X(X^TX)^{-1}X^T$ be the hat matrix and suppose $P = \left[\ P_1 \mid P_2 \mid \cdots \mid P_n\ \right]$ is the orthogonal matrix which has columns which are the eigenvectors of H with the first being those which are non-zero. Let $\mathbf{v} = P^T\mathbf{y}$.

   (a) Give expressions for $\mathbf{y}$ and $H\mathbf{y}$ in terms of the eigenvectors, $P_1, \cdots, P_n$ and $\mathbf{v}$, assuming that the number of predictors is $k+1$ and X is full rank $k+1$.

   Solution: From the definition,

   $$\mathbf{y} = P\mathbf{v} = \sum_{i=1}^{n} v_i P_i$$

   .

   Since H is idempotent it has eigenvalues which are 0 or 1 and has rank $k+1$ because $Hy = Xb$ and is thus in the column space of X. Hence there are $k+1$ eigenvalues which are 1, and $H = P\Lambda P^T$ where Λ is a diagonal matrix with the top left of the diagonal having $k+1$ 1's and the remaining $n-k-1$ entries all 0. So

   $$\begin{aligned} H\mathbf{y} &= P\Lambda P^T P\mathbf{v} \\ &= P\Lambda\mathbf{v} \\ &= \sum_{i=1}^{k+1} v_i P_i \end{aligned}$$

   because $\Lambda\mathbf{v}$ is a column vector of $k+1$ $x-$values followed by $n-k-1$ 0's.

   (b) Hence show that the data vector y is orthogonal to the least-squares fitted values $\hat{\mathbf{y}} = X\mathbf{b}$ if, and only if, the fitted values are 0.

   Solution: Since $H\mathbf{y} = X\mathbf{b}$, the dot product of $\mathbf{y}$ with the fitted values is

   $$\begin{aligned} \mathbf{y}^T H\mathbf{y} &= \left(\sum_{i=1}^{n} v_i P_i\right)^T \sum_{i=1}^{k+1} v_i P_i \\ &= \sum_{i=1}^{k+1} v_i^2 P_i^T P_i + \sum_{i,j} v_i x_j P_i P_j \\ &= \sum_{i=1}^{k+1} v_i^2 \end{aligned}$$

   where the sum over i, j is for $i \neq j, i = 1, \cdots, n, j = 1, \cdots, k+1$ and the last equality follows since the vectors P_i are othogonal and have unit length. Hence, $\mathbf{y}^T H\mathbf{y} = 0$ if, and only if, $\sum_{i=1}^{k+1} v_i^2 = 0$ which occurs, if, and only if, $x_1 = \cdots = x_{k+1} = 0$. The latter is the same as the fitted values $X\mathbf{b} = H\mathbf{y} = \sum_{i=1}^{k+1} v_i P_i = \mathbf{0}$.

   Note that in these circumstances $\mathbf{y}$ is free to vary in the subspace generated by $P_{k+2}, \cdots, P_n$. In Figure 1 of Module 4, this means that $\mathbf{y}$ lies in a subspace of dimension 1 ie it has one degree of freedom to vary. Further, in this case $\mathbf{y} = (I-H)\mathbf{y} = \mathbf{e} =$ is orthogonal to the whole of the column space of X.

(c) Suppose

$$X = \begin{bmatrix} 1 & 2 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}.$$

Find all data vectors $\mathbf{y}$ for which the fitted values are 0.

Solution: From the previous part of this question, the data vectors are those which are orthogonal to the plane spanned by the two columns of X. The data vector $\mathbf{y} = \begin{bmatrix} 0 & 1 & -1 \end{bmatrix}^T$ is orthogonal to both columns of X and generates the orthogonal complement of the column space of X. All the other vectors are multiples of this one.

2. Michael asked some questions about leverage which motivated this question.

(a) Suppose that H is an idempotent and symmetric matrix. Show that the diagonal entries of H are in $[0, 1]$ and that their sum is the rank of H.

Solution: Since H is symmetric, $H_{ii}^2 = (H_{ii})^2 + \sum_{j \neq i} H_{ij}H_{ji} = (H_{ii})^2 + \sum_{j \neq i}(H_{ij})^2 \geq 0$.

Because H is idempotent, so $H^2 = H$ gives $H_{ii} \geq 0$.

In the first Workshop, we showed that $I - H$ is also idempotent and symmetric. Hence its diagonal entries, $1 - H_{ii} \geq 0$, giving $H_{ii} \leq 1$.

Since H is idempotent and symmetric, its rank is the same as its trace.

(b) Interpret part (a) in terms of leverage.

Solution: Since the hat matrix $H = X(X^TX)^{-1}X^T$ is symmetric and idempotent and its diagonal entries are the leverages, part (a) tells us that the leverages are between 0 and 1. The sum of the leverages is the rank of H. Part 1(a) shows that the rank of H is $k+1$ so the sum of the leverages is $k+1$.

(c) The formula for the variance of the ith residual is var $(\varepsilon) = \sigma^2(1 - H_{ii})$. If observation i has a leverage close to 1, how does this show that the response variable, y, for a row of the X matrix with large leverage *can* have a significant effect on the model parameters? When does such an observation have a significant effect?

Solution: If observation i has a leverage close to 1, then the variance of the residual is close to 0. This implies that the fitted value must be close to the response value, so the fits must pay a lot of attention to the ith response variable. Cook's distance is a measure of the difference between the parameters with and without the ith observation. It is proportional to the product of the square of the estimated standardised residual and the leverage, by the formula from lectures. A combination of them determines when the effect is significant.

There is a good demonstration of this available in Mathematica. The demonstration allows you to alter one point dynamically and see the effect on the least squares line. The demonstration is available at the following web address: Influential Points in Regression.

In Lab Exercise 2 below this is explored with two fixed points added to a random scatter of bivariate normal random variables .

3. Consider the dataset from the Week 3 lab. As before, do the following using both matrix calculations and R's `lm` commands.

(a) Calculate 95% confidence intervals for the model parameters.

Solution:

```
y <- c(8, 15, 16, 20, 25, 40)
X <- cbind(rep(1, 6), c(8, 12, 14, 16, 16, 20))
b <- solve(t(X) %*% X, t(X) %*% y)
e <- y - X %*% b
SSRes <- sum(e^2)
n <- length(y)
p <- dim(X)[2]
s <- sqrt(SSRes/(n-p))
C <- solve(t(X)%*%X)
c(b[1] - qt(0.975,df=n-p)*s*sqrt(C[1,1]), b[1] + qt(0.975,df=n-p)*s*sqrt(C[1,1]))
```

```
## [1] -35.042213   3.906213
```

```
c(b[2] - qt(0.975,df=n-p)*s*sqrt(C[2,2]), b[2] + qt(0.975,df=n-p)*s*sqrt(C[2,2]))
```

```
## [1] 1.213055 3.842945
```

Alternatively,

```
income <- data.frame(income=y,education=X[,2])
model <- lm(income ~ education, data=income)
confint(model, level=0.95)
```

```
##                   2.5 %   97.5 %
## (Intercept) -35.042213 3.906213
## education     1.213055 3.842945
```

(b) Calculate a 95% confidence interval for the average income of a person who has had 18 years of formal education.
Solution:

```
xst <- c(1,18)
hw <- qt(0.975, df=n-p)*s*sqrt(t(xst) %*% C %*% xst)
c(t(xst) %*% b - hw, t(xst) %*% b + hw)
```

```
## [1] 23.0613 36.8107
```

Alternatively,

```
person <- data.frame(education=18)
predict(model, person, interval="confidence", level=0.95)
```

```
##      fit     lwr     upr
## 1 29.936 23.0613 36.8107
```

(c) Calculate a 95% prediction interval for the income of a single person who has had 18 years of formal education.
Solution:

```
xst <- c(1,18)
hw <- qt(0.975, df=n-p)*s*sqrt(1+t(xst) %*% C %*% xst)
c(t(xst) %*% b - hw, t(xst) %*% b + hw)
```

```
## [1] 16.10301 43.76899
```

Alternatively,

```
person <- data.frame(education=18)
predict(model, person, interval="prediction", level=0.95)
```

```
##      fit      lwr      upr
## 1 29.936 16.10301 43.76899
```

4. For simple linear regression, $y = \beta_0 + \beta_1 x + \varepsilon$, show that a $100(1-\alpha)\%$ confidence interval for the mean response when $x = x^*$ can be written as

$$b_0 + b_1 x^* \pm t_{\alpha/2} s \sqrt{\frac{1}{n} + \frac{(x^* - \bar{x})^2}{s_{xx}}}$$

where $s_{xx} = \sum_{i=1}^{n} v_i^2 - n\bar{x}^2$.
Similarly, show that a $100(1-\alpha)\%$ prediction interval for a new response when $x = x^*$ can be written as

$$b_0 + b_1 x^* \pm t_{\alpha/2} s \sqrt{1 + \frac{1}{n} + \frac{(x^* - \bar{x})^2}{s_{xx}}}.$$

Solution: For simple linear regression we have

$$\begin{aligned} X^TX &= \begin{bmatrix} 1 & 1 & \dots & 1 \\ ve_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \\ &= \begin{bmatrix} n & \sum_{i=1}^n v_i \\ \sum_{i=1}^n v_i & \sum_{i=1}^n v_i^2 \end{bmatrix} \\ (X^TX)^{-1} &= \frac{1}{n\sum_{i=1}^n v_i^2 - \left(\sum_{i=1}^n v_i\right)^2} \begin{bmatrix} \sum_{i=1}^n v_i^2 & -\sum_{i=1}^n v_i \\ -\sum_{i=1}^n v_i & n \end{bmatrix}. \end{aligned}$$

Let $\mathbf{t} = [1\ x^*]^T$. Then our CI for the mean response when $x = x^*$ is

$$\mathbf{t}^T\mathbf{b} \pm t_{\alpha/2} s\sqrt{\mathbf{t}^T(X^TX)^{-1}\mathbf{t}} = b_0 + b_1x^* \pm t_{\alpha/2} s\sqrt{\mathbf{t}^T(X^TX)^{-1}\mathbf{t}}.$$

The term in the square root is

$$\begin{aligned} \mathbf{t}^T(X^TX)^{-1}\mathbf{t} &= \frac{1}{n\sum_{i=1}^n v_i^2 - \left(\sum_{i=1}^n v_i\right)^2} \begin{bmatrix} 1 & x^* \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n v_i^2 & -\sum_{i=1}^n v_i \\ -\sum_{i=1}^n v_i & n \end{bmatrix} \begin{bmatrix} 1 \\ ve^* \end{bmatrix} \\ &= \frac{1}{ns_{xx}} \begin{bmatrix} \sum_{i=1}^n v_i^2 - x^*\sum_{i=1}^n v_i & -\sum_{i=1}^n v_i + nx^* \end{bmatrix} \begin{bmatrix} 1 \\ ve^* \end{bmatrix} \\ &= \frac{1}{ns_{xx}}[\textstyle\sum_{i=1}^n v_i^2 - 2x^*\sum_{i=1}^n v_i + n(x^*)^2] \\ &= \frac{1}{ns_{xx}}[\textstyle\sum_{i=1}^n v_i^2 - n\bar{x}^2 + n\bar{x}^2 - 2nx^*\bar{x} + n(x^*)^2] \\ &= \frac{1}{ns_{xx}}[s_{xx} + n(x^* - \bar{x})^2] \\ &= \frac{1}{n} + \frac{(x^* - \bar{x})^2}{s_{xx}} \end{aligned}$$

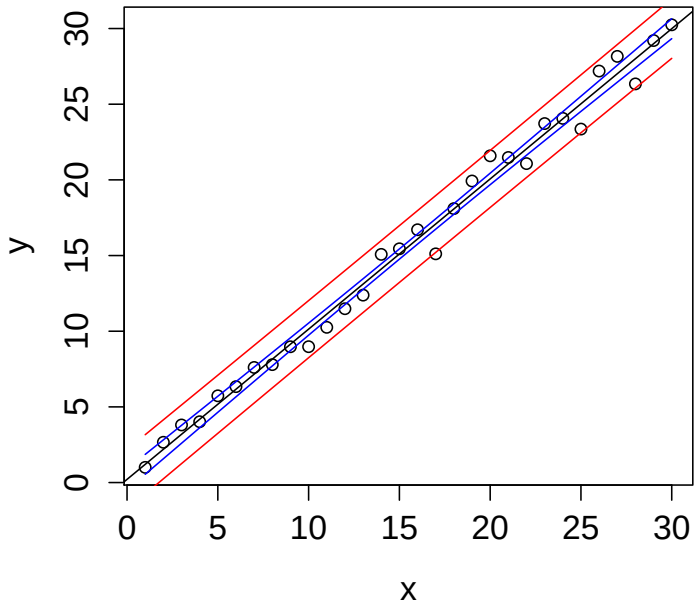
as required.

The proof for the prediction interval is similar.

5. We can generate some data for a simple linear regression as follows:

```
n <- 30
x <- 1:n
y <- x + rnorm(n)
```

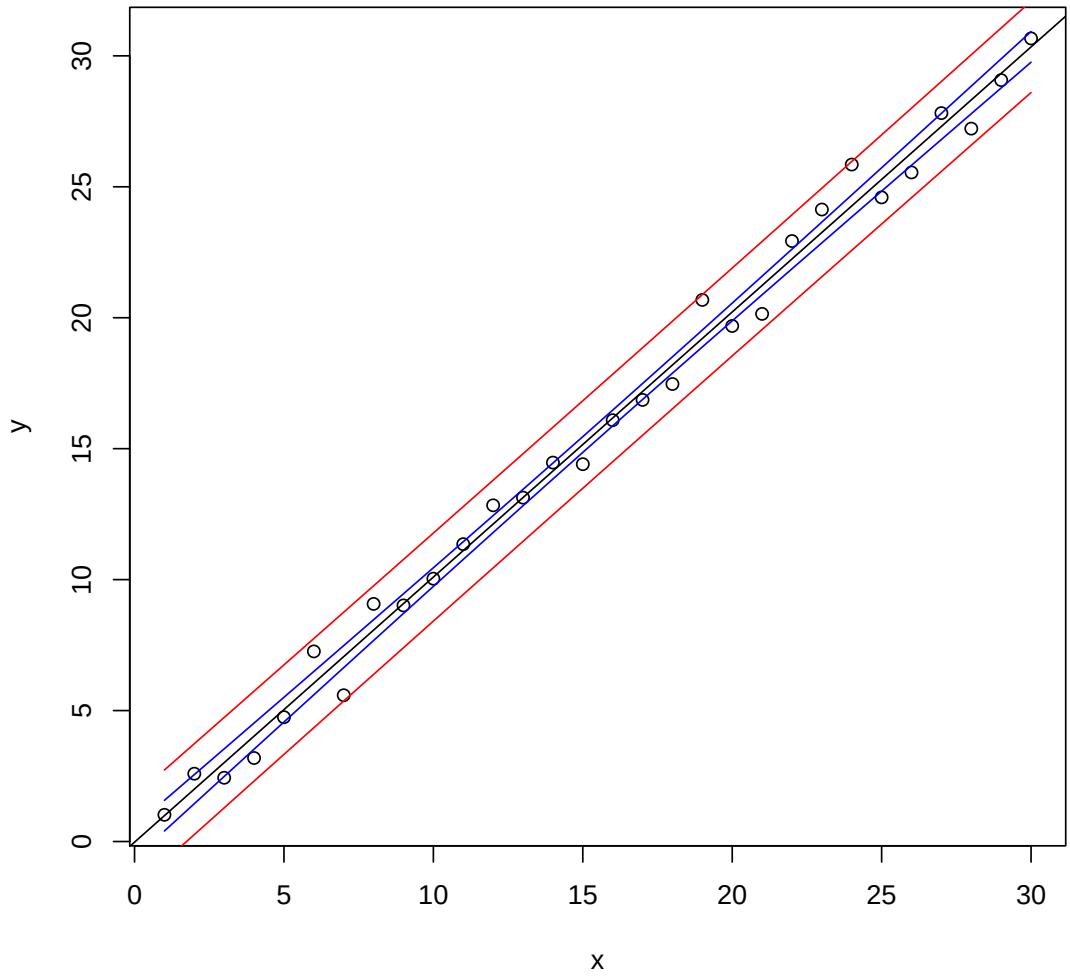
Construct 95% CI's for $\mathbb{E}[y]$ and 95% PI's for y, when $x = 1, 2, \dots, n$. Join them up and plot them on a graph of y against x. Your plot should look something like this:



What proportion of the y's should you expect to lie beyond the outer lines?

Solution:

```
fit <- lm(y ~ x)
newdata <- data.frame(x=x)
CI_u <- predict(fit,newdata,interval='confidence',level=0.95)[,3]
CI_l <- predict(fit,newdata,interval='confidence',level=0.95)[,2]
PI_u <- predict(fit,newdata,interval='prediction',level=0.95)[,3]
PI_l <- predict(fit,newdata,interval='prediction',level=0.95)[,2]
plot(x, y)
abline(fit$coeff[1], fit$coeff[2])
lines(x, CI_u, col="blue")
lines(x, CI_l, col="blue")
lines(x, PI_u, col="red")
lines(x, PI_l, col="red")
```



We expect around 5% of the y's to lie beyond the PI's.

6. In this exercise, we look at the dangers of overfitting. Generate some observations from a simple linear regression:

```
set.seed(3)
X <- cbind(rep(1,100), 1:100)
beta <- c(0, 1)
y <- X %*% beta + rnorm(100)
```

Put aside some of the data for testing and some for fitting:

```
Xfit <- X[1:50,]
yfit <- y[1:50]
Xtest <- X[51:100,]
ytest <- y[51:100]
```

(a) Using only the fitting data, estimate $\boldsymbol{\beta}$ and hence the residual sum of squares. Also calculate the residual sum of squares for the test data, that is, $\sum_{i=51}^{100}(y_i - b_0 - b_1 v_i)^2$.
Solution:

```
betafit <- solve(t(Xfit)%*%Xfit, t(Xfit)%*%yfit)
( SSfit <- sum((yfit - Xfit%*%betafit)^2) )

## [1] 38.16794

( SStest <- sum((ytest - Xtest%*%betafit)^2) )

## [1] 36.22107
```

Now add 10 extra predictor variables which we know have nothing to do with the response:

```
X <- cbind(X, matrix(runif(1000), 100, 10))
Xtest <- X[51:100,]
Xfit <- X[1:50,]
```

Again using only the fitting data, fit the linear model $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, and show that the residual sum of squares has reduced (this has to happen). Then show that the residual sum of squares for the test data has gone up (this happens most of the time).

Explain what is going on.

Solution:

```
( betafit <- solve(t(Xfit)%*%Xfit, t(Xfit)%*%yfit) )

##              [,1]
##  [1,] -0.63651872
##  [2,]  1.01113287
##  [3,]  0.06578001
##  [4,]  0.60104434
##  [5,]  0.29716283
##  [6,]  0.22470900
##  [7,] -0.40165740
##  [8,]  0.39186181
##  [9,] -0.02118371
## [10,] -0.34238715
## [11,] -0.53698933
## [12,]  0.12811422

( SSfit <- sum((yfit - Xfit%*%betafit)^2) )

## [1] 34.27347

( SStest <- sum((ytest - Xtest%*%betafit)^2) )

## [1] 39.07738
```

With the training data, we can match some of the noise (the $\boldsymbol{\varepsilon}$ term) using the new predictor variables. However, this is of no use when applied to the test data, as there is no systematic relationship between the noise and the new variables.

(b) Repeat the above, but this time add x^2, x^3 and x^4 terms:

```
X <- cbind(X[, 1:2], (1:100)^2, (1:100)^3, (1:100)^4)
```

Solution:

```
Xfit <- X[1:50,]
Xtest <- X[51:100,]

( betafit <- solve(t(Xfit)%*%Xfit, t(Xfit)%*%yfit) )

##               [,1]
## [1,] -5.605561e-01
```

```
## [2,]  1.124641e+00
## [3,] -1.252249e-02
## [4,]  4.589517e-04
## [5,] -5.217886e-06

( SSfit <- sum((yfit - Xfit%*%betafit)^2) )

## [1] 34.94215

( SStest <- sum((ytest - Xtest%*%betafit)^2) )

## [1] 270310.6
```

Here the fit for the test data is absolutely horrendous. The problem is that the term $\beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4$ is small for the training data, where it is fitting the noise term $\boldsymbol{\varepsilon}$, but for the test data this term becomes very large, resulting in a very poor fit. So overfitting is generally a bad thing, but overfitting with polynomials can be a very bad thing, in particular when you try and apply the fit beyond the range of the fitting data.

Lab questions

1. What will be the output of the following code? Try to answer this without typing it up.

```
fb <- function(n) {
    if (n == 1 || n == 2) {
        return(1)
    } else {
        return(fb(n - 1) + fb(n - 2))
    }
}
fb(8)
```

Solution: `fb(n)` returns the n-th Fibonacci number, so `fb(8)` returns 21.

2. This question writes a function to calculate leverages and generate histograms for them, and applies it to three different datasets in order to better understand leverage.

   (a) In R, generate 20 observations from a standard bivariate normal distribution with correlation 0.8. (You can do this using the definition and pairs of standard independent normal random variables or using the function `rmvnorm` in the package `mvtnorm`). The three data sets are pairs $(x1, y1), (x2, y2), (x3, y3)$ with the first data set having 20 pairs of (x, y) and the second two having 21. Specifically, the three data sets are:

      i. The 20 bivariate observations, labelling the first of each pair as `x1` in R, and the second as `y1`.

      ii. New variables, `x2`, `y2`, which include the pairs (`x1`,`y1`) together with an additional point $(10, 10)$.

      iii. New variables, `x3`, `y3`, which include pairs in (`x1`,`y1`) together with an additional point $(10, -0.3)$.

   Solution:

```
x <- rmvnorm(20, mean=c(0,0),
sigma=matrix(c(1,0.7,0.7,1),ncol=2))
x1 <- x[,1]
y1 <- x[,2]
x2 <- c(x1,10)
x3 <- x2
y2 <- c(x1,10)
y3 <- c(x1,-0.3)
```

(b) Fit simple linear regression models of the response $y$to the predictor x for each of the three data sets in part (a), storing the results in `model1`, `model2`, `model3`, including the options `x=TRUE` in order that it is possible to access the X matrix of the models subsequently.

Solution:

```
model1 <- lm(y1 ~ x1,x=TRUE)
model2 <- lm(y2 ~ x2,x=TRUE)
model3 <- lm(y3 ~ x3,x=TRUE)
```

(c) Write a function, `histhat`, with argument, `model`, that uses matrix operations to find the leverage values in model and prints them out with the model formula as heading, as well as generates a histogram of the leverage values. To label the printout and the histogram, the following R commands may be helpful:

i. `writeLines`
ii. `noquote` to get text strings without quotes printed out
iii. `paste` including the text string `"\n"` to get a new line before a subsequent `print`

Solution:

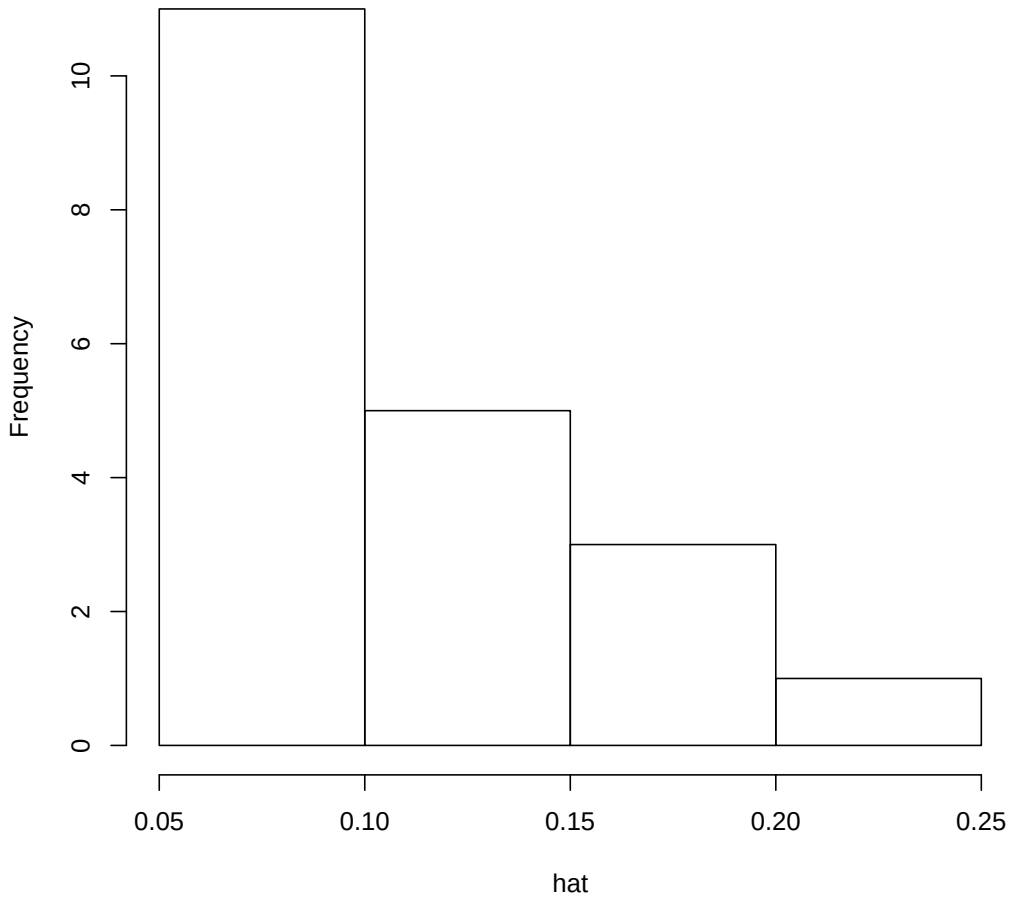
```
histhat <- function(model){
Xmat <- model$x
XX <- t(Xmat)%*%Xmat
H <- Xmat%*%solve(XX)%*%t(Xmat)
hat <- diag(H)
writeLines(
noquote(
paste("Leverages for model",model$call["formula"],"\n")
)
)
print(hat)
hist(hat,
main = paste("Leverages for model",model$call["formula"])
)
}
```

(d) Generate the printouts of leverages and histogram of leverages for the three models

Solution:

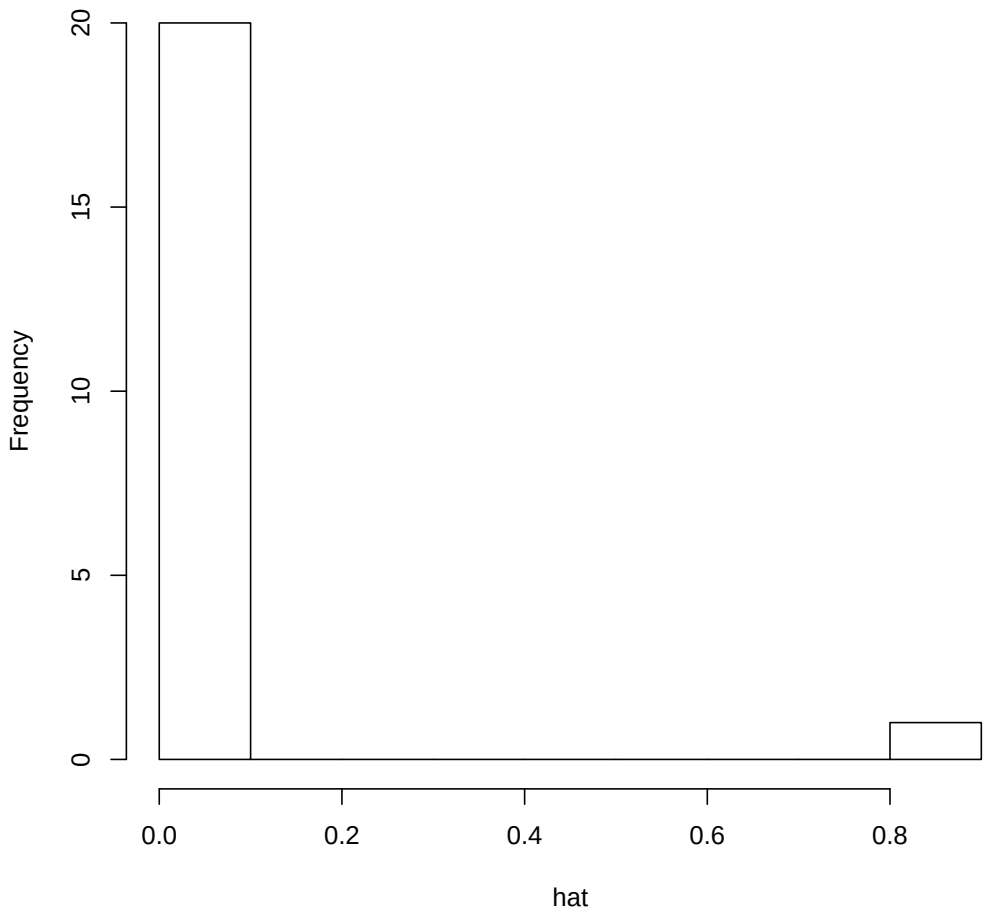
```
histhat(model1)
```

```
## Leverages for model y1 ~ x1
##
##          1          2          3          4          5          6
## 0.05082128 0.05875790 0.13381598 0.17643250 0.11626359 0.05940889
##          7          8          9         10         11         12
## 0.17586485 0.13573954 0.06235337 0.05572307 0.05419170 0.05179270
##         13         14         15         16         17         18
## 0.11022444 0.15661734 0.05131934 0.22788965 0.05917225 0.12688354
##         19         20
## 0.05196275 0.08476531
```



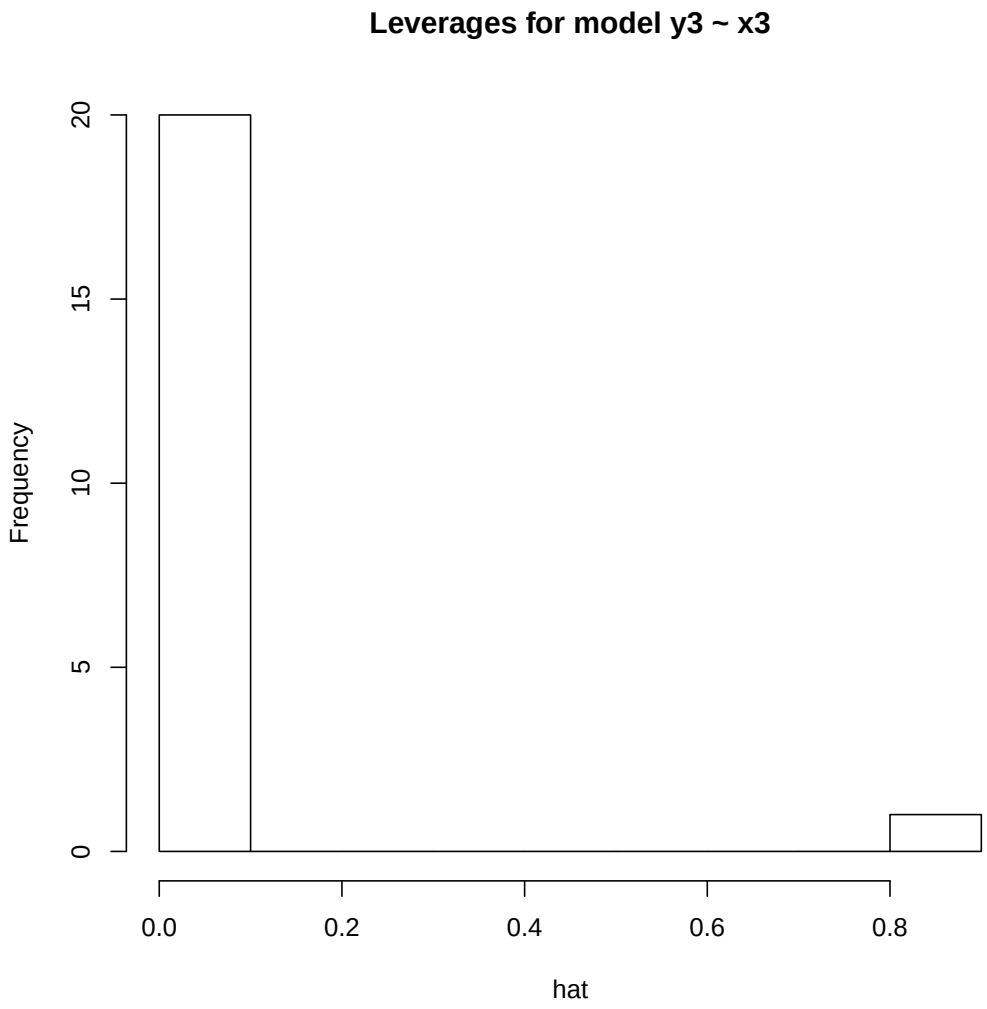

```
histhat(model2)
```

```
## Leverages for model y2 ~ x2
##
##          1          2          3          4          5          6
## 0.04866405 0.05458737 0.07499558 0.08490734 0.05163785 0.04763296
##          7          8          9         10         11         12
## 0.05846242 0.07545488 0.04762384 0.05339092 0.04791582 0.05146733
##         13         14         15         16         17         18
## 0.05102592 0.05614023 0.04846698 0.09637189 0.04763706 0.07332818
##         19         20         21
## 0.04828366 0.04875480 0.83325092
```




```
histhat(model3)
```

```
## Leverages for model y3 ~ x3
##
##          1          2          3          4          5          6
## 0.04866405 0.05458737 0.07499558 0.08490734 0.05163785 0.04763296
##          7          8          9         10         11         12
## 0.05846242 0.07545488 0.04762384 0.05339092 0.04791582 0.05146733
##         13         14         15         16         17         18
## 0.05102592 0.05614023 0.04846698 0.09637189 0.04763706 0.07332818
##         19         20         21
## 0.04828366 0.04875480 0.83325092
```



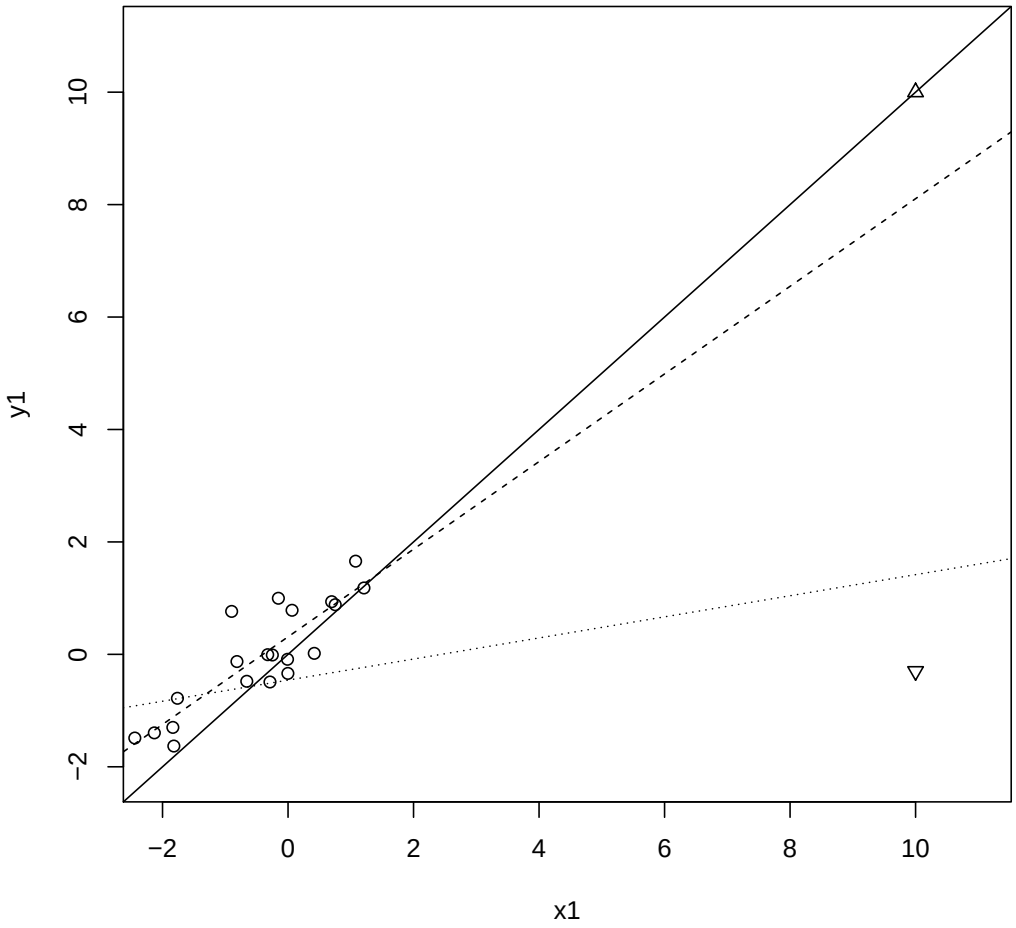
(e) Comment on the leverage outputs.

Solution:

(f) Plot the data in `y1` versus `x1` using the options `xlim` and `ylim` to make sure that the axes go from -2.1 to 11. Add the point $(10, 10)$ with an upper triangle and point $(10, -0.3)$ with a lower triangle. Add the lines from `model2` and `model3` using dashed and dotted lines. Add the line from `model1` in a solid line using the intercept and slope inputs.

Solution:

```
plot(x1,y1,xlim=c(-2.1,11),ylim=c(-2.1,11))
points(x=x2[21],y=y2[21],pch=2)
points(x=x3[21],y=y3[21],pch=6)
abline(model3,lty="dotted")
abline(model2)
abline(a=model1$coef[1],b=model1$coef[2],lty="dashed")
```



(g) Comment on the plots.

3. Suppose you needed all 2^n binary sequences of length n. One way of generating them is with nested for loops. For example, the following code generates a matrix `binseq`, where each row is a different binary sequence of length three.

```
binseq <- matrix(nrow = 8, ncol = 3)
r <- 0 # current row of binseq
for (i in 0:1) {
  for (j in 0:1) {
    for (k in 0:1) {
      r <- r + 1
      binseq[r,] <- c(i, j, k)
    }
  }
}
binseq
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    1
## [3,]    0    1    0
## [4,]    0    1    1
```

```
## [5,]    1    0    0
## [6,]    1    0    1
## [7,]    1    1    0
## [8,]    1    1    1
```

Clearly this approach will get a little tedious for large n. An alternative is to use recursion. Suppose that A is a matrix of size $2^n \times n$, where each row is a different binary sequence of length n. Then the following matrix contains all binary sequences of length $n+1$:

$$\left(\begin{array}{c|c} \mathbf{0} & A \\ \hline \mathbf{1} & A \end{array}\right).$$

Here $\mathbf{0}$ is a vector of zeros and $\mathbf{1}$ is a vector of ones.

Use this idea to write a recursive function `binseq`, which takes as input an integer n and returns a matrix containing all binary sequences of length n, as rows of the matrix. You should find the functions `cbind` and `rbind` particularly useful.

Solution:

```
binseq <- function(n) {
  # all binary sequences of length n, where n is a +ve integer
  if (n == 1) {
    A <- matrix(0:1, nrow=2, ncol=1)
    return(A)
  } else {
    A <- binseq(n-1)
    B <- rbind(cbind(0, A), cbind(1, A))
    return(B)
  }
}
binseq(3)

##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    1
## [3,]    0    1    0
## [4,]    0    1    1
## [5,]    1    0    0
## [6,]    1    0    1
## [7,]    1    1    0
## [8,]    1    1    1
```

4. Let $A = (a_{i,j})_{i,j=1}^n$ be a square matrix, and denote by $A_{(-i,-j)}$ the matrix with row i and column j removed. If A is a 1×1 matrix then $det(A)$, the determinant of A, is just $a_{1,1}$. For $n \times n$ matrices we have, for any i,

$$det(A) = \sum_{j=1}^{n} (-1)^{i+j} \, a_{i,j} \, det(A_{(-i,-j)}).$$

Use this to write a *recursive* function to calculate $det(A)$.

Solution:

```
my_det <- function(X) {
  if (length(X) == 1) {
    return(X)
  } else {
    S <- 0
```

```
    for (i in 1:dim(X)[1]) {
      S <- S + X[1, i]*(-1)^(1+i)*my_det(X[-1, -i])
    }
    return(S)
  }
}

X <- matrix(runif(25), nrow=5, ncol=5)
my_det(X)

## [1] 0.06332114

det(X) # R's built in version

## [1] 0.06332114
```