Problem Set 3

INF 511

Purnabhishek Sripathi

1 Exploring uncertainty in $\hat{B}$

Here is a simulated data set for simple linear regression.

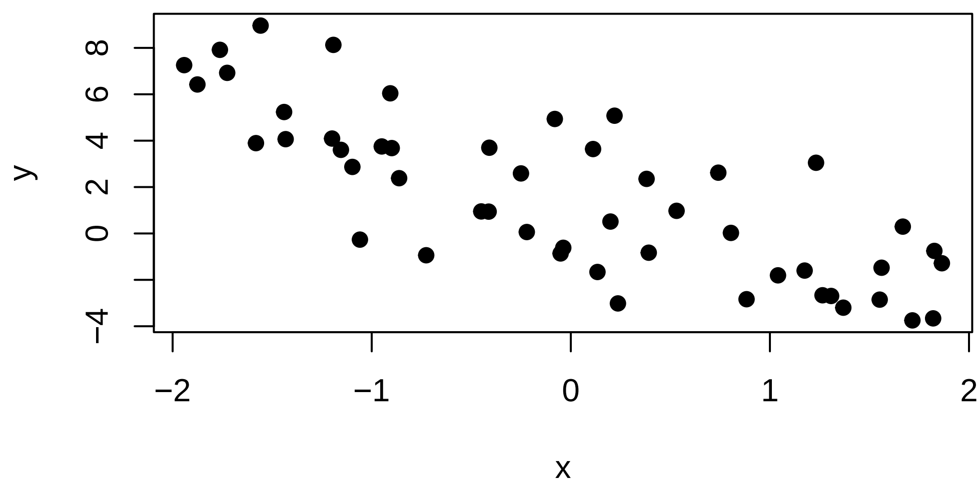
```
set.seed(5)

# Model parameters
betas = c(1.5, -2.7)
sigma = 2.2
n = 50

# Input variable
x = runif(n, min = -2, max = 2)
xmat = cbind(1, x)

# Outcome variable
y = xmat %*% betas + rnorm(n, 0, sigma)

# plot
plot(y ~ x, pch = 19)
```



1.1 Run the `lm()` function

Conduct a linear regression analysis on this data set using least squares via the `lm()`.

- Create a data frame to store your y and x variable.

```r
# creating the dataframe
#to store the x and y variables

data_frame <- data.frame(y=y, x=x)
data_frame
```

```
             y           x
1   4.09202385 -1.19914219
2   2.62053513  0.74087438
3   0.29504406  1.66750310
4   2.38290525 -0.86240217
5   3.89362865 -1.58139949
6   0.02359252  0.80422984
7   3.63935934  0.11183994
8   3.04831309  1.23174080
9  -0.75257335  1.82600050
10  8.96139533 -1.55818793
11  6.04198498 -0.90686020
12 -0.61851440 -0.03794719
13 -0.93980343 -0.72638393
14 -3.01587545  0.23669131
15  3.75025622 -0.94962745
16  8.12988053 -1.19249916
17  0.94939093 -0.44989701
18 -2.85306716  1.55147909
19  5.07730592  0.21969022
20 -3.19998903  1.36871754
21 -1.47734544  1.56082846
22 -2.83498921  0.88280388
23  3.60498698 -1.15463891
24  2.86873059 -1.09713084
25  5.23591153 -1.44006521
26  4.93607721 -0.08034458
27  2.58894933 -0.25035256
28 -1.28396241  1.86385656
29  4.06529313 -1.43234436
30 -3.66061344  1.81991555
31  0.06201321 -0.22107778
32  7.91615584 -1.76255011
33  3.68064885 -0.89940326
34  6.42259876 -1.87540696
35  7.25679633 -1.94207062
36 -0.86190768 -0.05132388
37  2.35298940  0.38060649
38 -0.82922772  0.39141215
39  3.69704582 -0.40927794
40  0.94210427 -0.41267841
41 -2.66467134  1.26430428
42 -0.26527061 -1.05920498
43 -2.69394368  1.30748098
44 -1.66078363  0.13390373
45 -3.74784675  1.71577085
46  0.51320286  0.19917890
```

```
47 -1.80589050  1.04057011
48  6.92355716 -1.72596095
49 -1.59977161  1.17443736
50  0.97549182  0.53121371
```

- Run the `lm()` function with an appropriate formula. Store the output in an object.

```
#running the lm() function on the above data_frame

result_data <- lm(y~x, data=data_frame)
result_data
```

```
Call:
lm(formula = y ~ x, data = data_frame)

Coefficients:
(Intercept)            x
      1.545       -2.355
```

- Extract and report the estimated coefficients from the `lm()` output object.

```
#fetching the coefficients of the result_data

coeff_of_result_data <- coef(result_data)
coeff_of_result_data
```

```
(Intercept)           x
   1.544624   -2.354843
```

1.2 Estimated residual variance

Remember from lecture that the estimated residual variance can be obtained from:

$$\hat{\sigma}^2 = \frac{1}{n-p}\hat{\epsilon}^T\hat{\epsilon}$$

, where $\hat{\epsilon}$ is the vector of estimated residuals. Below, create a function in R to compute the $\hat{\sigma}^2$.

```
n <- length(y)
p <- length((coeff_of_result_data))
n
```

```
[1] 50
```

```
p
```

```
[1] 2
```

- Extract the estimated residuals from the `lm()` output.

```
#fetching th eestimated residuls of result_data

residuals_of_result_data <- resid(result_data)
residuals_of_result_data
```

```
          1           2           3           4           5           6 
-0.27639156  2.82055381  2.67712771 -1.19254034 -1.37494261  0.37280334 
          7           8           9          10          11          12 
 2.35810078  4.40424503  2.00274677  3.74748366  2.36184771 -2.25249811 
         13          14          15          16          17          18 
-4.19494743 -4.00312866 -0.03059118  3.77710843 -1.65466985 -0.74420182 
         19          20          21          22          23          24 
 4.05001781 -1.52149840  0.65353620 -2.30074887 -0.65863019 -1.25946411 
         25          26          27          28          29          30 
 0.30016028  3.20225431  0.45478436  1.56050277 -0.85227672 -0.91962241 
         31          32          33          34          35          36 
-2.00321424  2.22100336  0.01807151  0.46168611  1.13890124 -2.52739140 
         37          38          39          40          41          42 
 1.70463383 -1.45213767  1.18863657 -1.57431256 -1.23205754 -4.30415589 
         43          44          45          46          47          48 
-1.15965551 -2.89008544 -1.25210012 -0.56238617 -0.90013549  1.31456638 
         49          50 
-0.37878028  0.68179257 
```

```r
#function to calculate the estimated residual variance!

residual_function <- function(residual_data, p)
{
  residual_length <- length(result_data)
  result_object <- sum(residual_data^2)/residual_length-p
  return (result_object)
}

#result_object
```

```r
#calculating the estimated_residual_variance

#estimated_residual_variance <- sum(residuals_of_result_data^2)
#result_object <- estimated_residual_variance/n-p
#result_object

result_object <- residual_function(residuals_of_result_data, p)
result_object
```

```
[1] 16.75296
```

- Create a function, which has two inputs: (1) the estimated residual vector $\hat{\epsilon}$, and (2) the number of model coefficients, p.
- Report your estimated $\hat{\sigma}^2$ by using your function, storing the output as an object, and printing the value of that object. Remember, to "print" the value of your object in the rendered PDF, you do not need to use the `print()` function.

1.3 Calculate the $SE(\hat{\beta}_i)$

- Calculate $(X^TX)^{-1}$, then
- Calculate the $SE(\hat{\beta}_i)$ for both the slope and intercept, using your calculations of $(X^TX)^{-1}$ and $\hat{\sigma}^2$, above.

```r
ones <- rep(1,n)
xmat <- cbind(ones, x)
#xmat

t_xmat <- t(xmat)
#t_xmat

#Calculating the $(X^TX)^{-1}$
xtran_inv <- solve(t_xmat %*% xmat)
xtran_inv
```

```
             ones            x
ones 0.0200095228 0.0003799603
x    0.0003799603 0.0151604710
```

```r
#calculating the covariane matrix to find the standard errors.
covariance_matrix <- xtran_inv * result_object
#covariance_matrix


#calculating the slope and intercept

slope <- sqrt(covariance_matrix[1,1])
intercept <- sqrt(covariance_matrix[2,2])

cat(paste("SE_intercept:", intercept, "\n"))
```

```
SE_intercept: 0.503967038189735
```

```r
cat(paste("SE_slope:", slope, "\n"))
```

```
SE_slope: 0.578980784517492
```

2 Confidence Intervals

2.1 Calculate the $t_{critical}$

Remember that to calculate confidence intervals for the slope and intercept, we need to calculate the critical t value from the appropriate t distribution with ν degrees of freedom. Calculate and report the $t_{critical}$ for the **90%** confidence interval.

```r
#calculating the T_critical value
degree_freedom <- n-p

t_critical <- qt(0.025, degree_freedom, lower.tail = FALSE)
t_critical
```

```
[1] 2.010635
```

2.2 Calculate the confidence intervals

For both slope and intercept, calculate the upper and lower bounds of the **90%** confidence interval, using the $t_{critical}$ and the $SE(\hat{\beta}_i)$ from above. Specifically, create a matrix or table to display: the estiamted coefficient from `lm()` output, as well as the upper and lower bounds of the **90%** confidence interval. Include both the slope and the intercept in the same matrix or table. You must use R to create the matrix or table.

```
interval = 1-0.90

t_critical_val <- qt(1-interval/2, degree_freedom)

val1 <- c(residuals_of_result_data[1]- t_critical_val* intercept)
val2 <- c(residuals_of_result_data[1] + t_critical_val * intercept)

intercept_ci <- c(val1, val2)
intercept_ci
```

```
        1          1
-1.1216573  0.5688742
```

```
valu1 <- c(residuals_of_result_data[2] - t_critical_val * slope)
valu2 <- c(residuals_of_result_data[2] + t_critical_val * slope)

slope_ci <-  c(valu1, valu2)
slope_ci
```

```
       2        2
1.849473 3.791634
```

```
#calculating the upper and lower bounds

name_coefficient <- c("Intercept", "Slope")
estimate_coefficient <- c(residuals_of_result_data[1], residuals_of_result_data[2])

lower_bound = c(intercept_ci[1], slope_ci[1])
upper_bound = c(intercept_ci[2], slope_ci[2])

confidence_inteval = data.frame(Coefficient = name_coefficient, Estimate = estimate_coefficient,
                      `Lower Bound` = lower_bound, `Upper Bound` = upper_bound)

confidence_inteval
```

```
  Coefficient   Estimate Lower.Bound Upper.Bound
1   Intercept -0.2763916   -1.121657   0.5688742
2       Slope  2.8205538    1.849473   3.7916344
```

2.3 Compare to `confint()`.

Now, use the `confint()` function to calculate the **90%** confidence intervals for the slope and intercept, and display the output.

```
#calculating the confidence interval
confi_inter <- confint(result_data, level = 0.9)
confi_inter
```

```
                  5 %      95 %
(Intercept)  1.030918  2.058330
x           -2.801992 -1.907694
```

3 Prediction intervals with `predict()`

We are going to use the `predict()` function to calculate the expected value of y and the prediction interval around this expectation, given the least squares analysis stored in the `lm()` function output.

```
#Predicting the expected value of y.
flag_val <- 0.55
expected_y <- predict(result_data, data = data.frame(x = flag_val))
expected_y
```

```
         1          2          3          4          5          6          7 
 4.3684154 -0.2000187 -2.3820837  3.5754456  5.2685713 -0.3492108  1.2812586 
         8          9         10         11         12         13         14 
-1.3559319 -2.7553201  5.2139117  3.6801373  1.6339837  3.2551440  0.9872532 
        15         16         17         18         19         20         21 
 3.7808474  4.3527721  2.6040608 -2.1088653  1.0272881 -1.6784906 -2.1308816 
        22         23         24         25         26         27         28 
-0.5342403  4.2636172  4.1281947  4.9357512  1.7338229  2.1341650 -2.8444652 
        29         30         31         32         33         34         35 
 4.9175698 -2.7409910  2.0652274  5.6951525  3.6625773  5.9609126  6.1178951 
        36         37         38         39         40         41         42 
 1.6654837  0.6483556  0.6229100  2.5084093  2.5164168 -1.4326138  4.0388853 
        43         44         45         46         47         48         49 
-1.5342882  1.2293018 -2.4957466  1.0755890 -0.9057550  5.6089908 -1.2209913 
        50 
 0.2936992 
```

3.1 Create a new data frame

Create a new data frame that stores a range of input variable x that we want to use to predict values of y. Complete the code chunk below.

```
n_pred = 100
new_df = data.frame(
  x = seq(from = -2, to = 2, length.out =  n_pred)
)
new_df
```

```
              x
1   -2.00000000
2   -1.95959596
3   -1.91919192
4   -1.87878788
5   -1.83838384
```

```
6   -1.79797980
7   -1.75757576
8   -1.71717172
9   -1.67676768
10  -1.63636364
11  -1.59595960
12  -1.55555556
13  -1.51515152
14  -1.47474747
15  -1.43434343
16  -1.39393939
17  -1.35353535
18  -1.31313131
19  -1.27272727
20  -1.23232323
21  -1.19191919
22  -1.15151515
23  -1.11111111
24  -1.07070707
25  -1.03030303
26  -0.98989899
27  -0.94949495
28  -0.90909091
29  -0.86868687
30  -0.82828283
31  -0.78787879
32  -0.74747475
33  -0.70707071
34  -0.66666667
35  -0.62626263
36  -0.58585859
37  -0.54545455
38  -0.50505051
39  -0.46464646
40  -0.42424242
41  -0.38383838
42  -0.34343434
43  -0.30303030
44  -0.26262626
45  -0.22222222
46  -0.18181818
47  -0.14141414
48  -0.10101010
49  -0.06060606
50  -0.02020202
51   0.02020202
52   0.06060606
53   0.10101010
54   0.14141414
55   0.18181818
56   0.22222222
57   0.26262626
58   0.30303030
59   0.34343434
```

```
60   0.38383838
61   0.42424242
62   0.46464646
63   0.50505051
64   0.54545455
65   0.58585859
66   0.62626263
67   0.66666667
68   0.70707071
69   0.74747475
70   0.78787879
71   0.82828283
72   0.86868687
73   0.90909091
74   0.94949495
75   0.98989899
76   1.03030303
77   1.07070707
78   1.11111111
79   1.15151515
80   1.19191919
81   1.23232323
82   1.27272727
83   1.31313131
84   1.35353535
85   1.39393939
86   1.43434343
87   1.47474747
88   1.51515152
89   1.55555556
90   1.59595960
91   1.63636364
92   1.67676768
93   1.71717172
94   1.75757576
95   1.79797980
96   1.83838384
97   1.87878788
98   1.91919192
99   1.95959596
100  2.00000000
```

3.2 Run the `predict()` function, including the `interval` and `level` options to specify the 90% prediction intervals.

Finish the code chunk below.

```
y_pred = predict(
  object = result_data,
  newdata = new_df,
  interval = "prediction",
  level = 0.9
)
```

3.3 Create the plot

The plot below shows the data points and the relationship between y and x. Add three lines to this plot, using the `lines()` function: (1) the expected value of y given your linear regression analysis, (2) the upper prediction interval, and (3) the lower prediction interval.

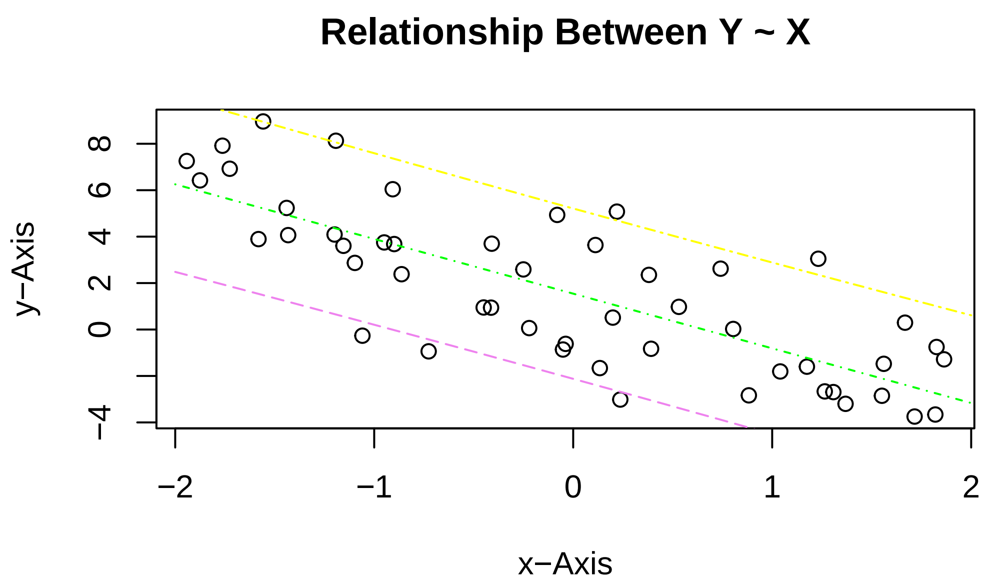
```
#plot(y ~ x, pch = 19)

#Plotting the data
plot(y~x, xlab = "x-Axis", ylab = "y-Axis", main = "Relationship Between Y ~ X")
#abline(result_object, col = "blue")

#Adding the three lines to the plot using lines()
lines(new_df$x, y_pred[, 1], col = "green", lty = 4)

lines(new_df$x, y_pred[, 2], col = "violet", lty = 5)

lines(new_df$x, y_pred[, 3], col = "yellow", lty = 6)
```



4 Rendering

Render this document as a `.PDF` file. Upload the rendered `.PDF` and the original `.QMD` onto BBLearn.